UNIVERSITY OF MAURITIUS

BSC SOFTWARE ENGINEERING
LEVEL I

SOFTWARE DESIGN FUNDAMENTALS
AND PROGRAMMING

HYPERDRIVE

3D CAR RACING GAME

ABHIJEET PITUMBUR	2014688
VIMAN SINGH SREEPAUL	2012071
OUDISH THACOOREE	2015177

ASSIGNMENT I

SIS10140Y

GROUP C

MAY 2021

Acknowledgement

It is a genuine privilege to express our special thanks of gratitude to our lecturers Mr. Somveer Kishnah and Mrs. Zarine Cadersaib who gave us huge knowledge and helped us to a great extent to complete this project. Their dedication, keen interest, timely advice and, above all, their impressive attitude to help their students, have been mainly responsible for finishing our work successfully.

Secondly, it is a pleasure to thank our fellow friends and seniors for their priceless suggestions and help, and awesome YouTube channels, mainly Imphenzia, Brackeys and Dimitris Lolis, for helping us in finalising this project within the limited time frame. We would like to thank all the content creators for making this material available for free.

This project was done using our knowledge of C# and C++. We prepared the numerous tasks to be completed effectively in order to finish the assignment on time. Throughout the process, we learnt a lot of new things, like better programming practices, and dealt with a variety of situations, for instance, team conflict resolution.

Abstract

This document presents Hyperdrive, an offline arcade-style 3D car racing video game. The project description, game design and implementation, the various challenges encountered during the development process, as well as the aims and objectives for choosing to develop this style of game, have been clearly stated. For a better understanding of the game, this document includes numerous tables and figures. Several tests were conducted to ensure that the application functions properly. Additionally, links to the various tutorials and assets that have been of great help during the development process have been included in the References section.

Hyperdrive can be downloaded for free at https://bit.do/hyperdrive

Table of Contents

List of Tables

List of Figures

1 - Introduction

1.1 - Introduction to Hyperdrive

Hyperdrive is an offline arcade-style 3D car racing video game. The game puts the player behind the wheel of sports cars from various manufacturers, from an entry-level model such as the Dodge Challenger, to a supercar like the Lamborghini Huracan. The player races against AI-controlled cars in a 3D environment, with the objective of winning the race and hence obtaining 3 stars in the level. Over the course of the game, the player can gradually unlock access to various new cars, levels, maps, and race modes. Players can also choose to play in a 1-player quick race or challenge a friend in the 2-player mode featured in Hyperdrive.

1.2 - Problem Statement

Video games have been around for decades, providing entertainment for children and adults alike. The video game sector is immensely large. In fact, it is larger than the movie and music industry combined, and it is only growing. The gaming industry's growth has resulted in a rise in the number of games that require a lot of storage and high-performance computers, especially those from big companies such as Electronic Arts (EA), Gameloft and Playground Games. In addition, many games need a significant amount of time to grind in order to progress, and they are considerably more sophisticated than early games. However, some car racing games do not offer their players the ability to play local multiplayer on a single device. Furthermore, many games are pay-to-win, that is, even if the game is free, the player must spend money on in-app purchases to progress significantly or even win, which removes the fun and struggle of unlocking new vehicles and levels by milling in the game.

1.3 - Aims and Objectives

1.3.1 - Aims

The aim is to create a game which is entertaining and challenging to play. The game implements an arcade-style racing to put fun and a fast-paced experience above all else, as the cars race in unique ways. It allows players to deal with situations of varying difficulties. Regardless of the fact that the game is designed to be both fun and challenging, it eliminates the need for high-performance computers in order to play high-quality games for free.

1.3.2 - Objectives

The objectives of the game are to:

- expose the player to a unique game,
- give the player a thrilling racing experience,
- enhance the hand-eye coordination, processing speed, spatial attention, tracking ability and decision-making skills of the player,
- generate interest of players in arcade-style games,
- run on a normal performance PC,
- provide players with a 2-player mode on the same device.

1.4 - Scope

The scope of the game is as follows:

- Hyperdrive is an offline arcade-style 3D car racing video game.
- The game includes 5 levels, each with different a map (Rural, Arctic, Desert, Urban, Moon) and race mode (Classic, Escape, Survival, Infection, Special).
- The main type of gameplay allows the player to race competitively and unlock new levels, maps, and cars.
- The secondary type of gameplay allows the player to race leisurely with any car in any level with any difficulty setting in any race mode with any number of laps against any number of racers, or even race against another player on the same computer with split-screen multiplayer.
- The game is not limited to adults only. Anyone above the age of 8 can play it.
- The codes are written in C#.

1.5 - Distribution of Task

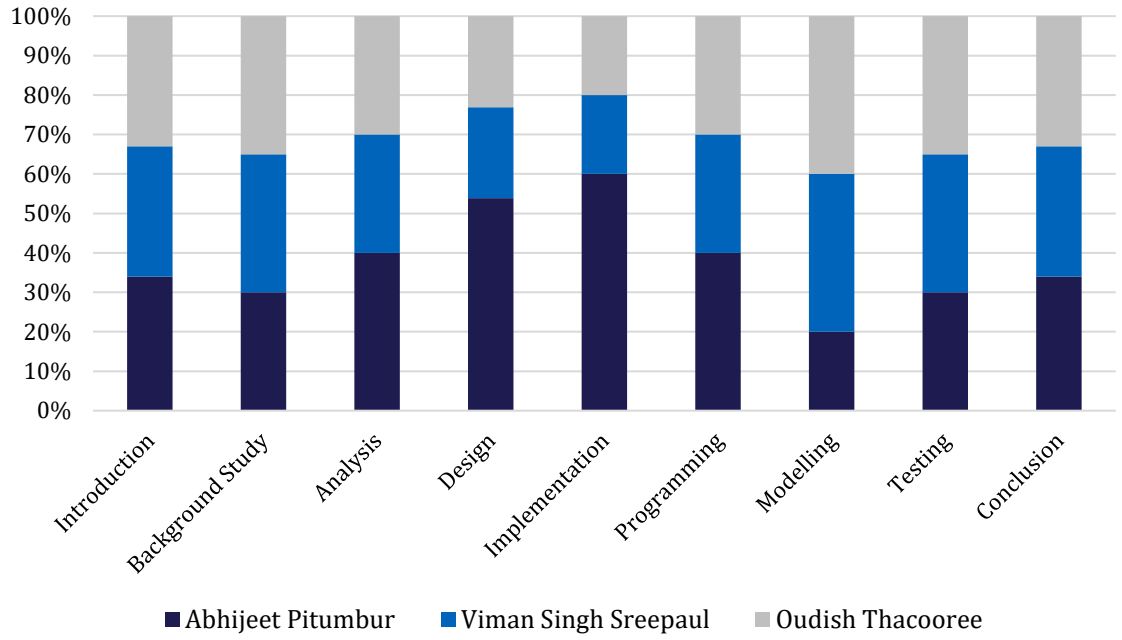


Table 1 - Distribution of Task

2 - Background Study

2.1 - Basic Concepts

Hyperdrive is an offline arcade-style 3D car racing video game. The main gameplay allows the player to choose one car, from a selection of 5, to race with. Depending on the selected level, the race occurs in one of 5 different race modes, namely Classic, Escape, Survival, Infection and Special.

The player races against AI-controlled cars, with the objective of winning the race and hence obtaining 3 stars in the level. The game grants 3 stars to the 1st-place racer, 2 stars to the 2nd-place racer, and 1 star to the 3rd-place racer. Depending on the level, new items can be unlocked with the player obtains 3 stars. Some race modes have a countdown timer, upon which is over, an action is triggered, such as elimination of the last-place racer or racers who have not completed the lap or infecting the last-place racer with a "contagious" temporary engine boost. Over the course of the game, the player can gradually unlock access to various new cars, levels, maps, and race modes, and eventually win the entire game.

Race modes:

- Classic - Cross the finish line first to win.
- Escape - Finish the laps before the timer runs out.
- Survival - Don't come last when the timer runs out.
- Infection - Infected cars have a temporary engine boost.
- Special - Mix of Escape and Survival modes.

The secondary gameplay allows the player to choose any level, car, race mode, difficulty, number of racers and number of laps. The player can even choose to race another player on the same computer with split-screen multiplayer.

2.2 - Existing Applications

2.2.1 - Sonic & Sega All-Stars Racing

Figure 1 - Sonic & Sega All-Stars Racing

Sonic & Sega All-Stars Racing is a 2010 kart racing video game, part of the Sega Superstars series, featuring characters from Sega franchises Players can choose to race as one of 20 characters from various Sega franchises such as Sonic the Hedgehog, Crazy Taxi, and Fantasy Zone. Characters race through various race circuits themed on different Sega franchises and collect power-ups to boost their speed or hamper their opponents.

Just like Sonic & Sega All-Stars Racing, Hyperdrive has cartoony graphics and is fun to play even for kids.

2.2.2 - ASPHALT 9 LEGENDS

Figure 2 - Asphalt 9 Legends

Asphalt 9 Legends is a racing video game developed by Gameloft Barcelona and published by Gameloft. Each of the cars belongs to a class progressively featuring higher performance and rarity: D, C, B, A, and S. The player starts with a car in the lowest class (Class D). All cars in the game now require "blueprints" to unlock and then subsequently "star up", with each of them having anywhere from 3 to 6 stars. For each new star, the car's performance capability is increased (at the cost of a lesser fuel tank quantity). Each car can also be customized with the new car editor feature.

Hyperdrive also contains some features offered in Asphalt 9 such as the feature to progressively unlock tracks and cars by finishing first in races and obtaining stars.

2.2.3 - NEED FOR SPEED PAYBACK

Figure 3 - Need for Speed Payback

Need for Speed Payback is a racing game set in an open world environment of Fortune Valley; a fictional version of Las Vegas, Nevada. It is focused on "action driving" and has three playable characters (each with different sets of skills) working together to pull off action movie like sequences. In contrast with the previous game, it also features a 24-hour day-night cycle. Unlike the 2015 Need for Speed reboot, Payback includes an offline single-player mode.

Need for Speed Payback features a total of 74 vehicles with downloadable contents. Toyota, Scion and Ferrari are not involved in the game due to licensing issues. However, the Subaru BRZ appears in the game. Aston Martin, Audi, Buick, Jaguar, Koenigsegg, Land Rover, Mercury, Pagani, and Plymouth make their return after their absence from the 2015 instalment, while Alfa Romeo, Infiniti, Mini and Pontiac were added via downloadable content.

Similarly to Need for Speed Payback, Hyperdrive also contains an option to play in an offline single player when the player can have fun racing against AI-driven cars.

2.2.4 - Hotshot Racing

Figure 4 - Hotshot Racing

Hotshot Racing's pick-up-and-play gameplay method and angular, low-polygon graphic look are reminiscent of arcade-style driving events from the 1990s. The game, developed by Lucky Mountain Games with assistance from Sumo Digital's racing masters, nails the retro-racing feeling in many ways. Although Hotshot Racing has the vivid colours of an Outrun game, it lacks the exhilarating level design that made many retro races so memorable. Hyperdrive's low-polygon design is inspired by Hotshot Racing.

2.3 - Potential Tools

There are hundreds of tools that can be used in a million different ways to create a game. Some vital types of tools are game engines, graphic editors, and code editors.

Game engines	Unity, Unreal Engine, Amazon Lumberyard, CryEngine, Godot
Graphic editors	Blender, Adobe Photoshop, SketchUp, Autodesk Maya, Daz Studio
Code editors	Microsoft Visual Studio, MonoDevelop, Microsoft Visual Studio Code, Adobe Brackets, Atom

2.4 - Comparative Analysis of Tools

2.4.1 - Game Engines

Unity is a cross-platform game engine that makes it simple to make interactive 3D content. Because of its excellent functionality, high-quality content, and ability to be used for any form of game, this gaming engine is the preferred option of many large organizations today. Unity is compatible with Windows, Mac, Linux, iOS, Android, and other platforms thanks to its all-in-one editor. The easy-to-use interface facilitates production and eliminates the need for instruction. The Unity Asset Store holds a constantly growing range of resources and content. Strengths: multi-platform support, good for teams of all sizes, 2D & 3D.

Figure 5 - Unity

Unreal Engine by Epic Games is one of the most popular and widely used game engines. It is basically a game development multi-platform engine for companies of all sizes that helps transform concepts into engaging visual content using real-time technology. Unreal Engine's strength is its ability to be customized to the point that games can be turned into truly unique experiences. This, however, necessitates the use of highly professional developers with extensive experience. Strengths: scalability, huge variety of features, rich customization abilities, 2D & 3D

Figure 6 - Unreal Engine

Amazon Lumberyard is a 3D game engine for creating games and fan communities. It comes with a virtual reality preview mode, visual scripting software, and Twitch integration. Lumberyard's integration with Amazon Web Services, a secure cloud platform designed and maintained by Amazon, makes it substantially easier to create games with online play. C++, P2P, and client topology are all well supported natively. Autodesk Maya and Adobe Photoshop are also supported by Lumberyard. Strengths: feature-rich platform, extensive capabilities.

Figure 7 - Amazon Lumberyard

CryEngine is a free-to-use platform that gives access to the entire engine source code as well as all engine functionality without the need to purchase a license. This is also a great way to get in-game assets, which can be found on the CryEngine Marketplace, which cuts down on time to market. CryEngine also offers a variety of free learning opportunities, though the utility of these is debatable. Ubisoft retains the Dunia Engine, an in-house adapted version of CryEngine from the original Far Cry, which is extensively used in their later versions of the iconic Far Cry series. Strengths: solid interface, stunning visual capabilities, impressive VR support.

Figure 8 - CryEngine

Godot is open-source and free to use under the MIT license. There are no hidden costs, royalties, or subscription fees. The Godot engine is ideal for both 2D and 3D game development. The engine includes a large number of popular resources, allowing us to concentrate on creating the game rather than reinventing the wheel. Strengths: easy to use, intuitive interface.

Figure 9 - Godot

2.4.2 - Graphic Editors

Blender is a free application for modeling, texturing, animation, and rendering. The open-source software has been around for a long time, and as a result, it has a devoted following of artists, educators, and enthusiasts. It has a powerful 3D modeling and sculpting toolkit and is widely regarded as a viable alternative to paid modeling software.

Figure 10 - Blender

Adobe Photoshop is a raster graphics editor developed and published by Adobe Inc. for Windows and macOS. Since then, the software has become the industry standard not only in raster graphics editing, but in digital art as a whole. With high familiarity from many its users, Photoshop is often part of the Adobe Creative Suite, which comes with a set of other useful tools.

Figure 11 - Adobe Photoshop

SketchUp describes itself as "the simplest free 3D modeling program on the internet - no strings attached". Its core 3D modeller runs right in the browser and comes with 10GB of storage, as well as user-generated and manufacturer-produced 3D models that can be imported for free into projects.

Figure 12 - SketchUp

Autodesk Maya is widely regarded as the industry standard for computer graphics, with an unrivalled collection of software and features. This massively extensible app is not for the faint of heart: its toolkit is extremely complicated and takes a long time to master. Particles, hair, solid body mechanics, fabric, fluid simulations, and character animation are just a few of Maya's many features. A subscription to Maya is not inexpensive, so this level of control comes at a cost.

Figure 13 - Autodesk Maya

Daz Studio, a 3D modeling software that used to cost almost $250, is now available for download for free. It is a 3D figure customization, posing, and animation tool that lets artists of all levels make digital art with virtual people, animals, props, vehicles, accessories, and environments.

Figure 14 - Daz Studio

2.4.3 - Code Editors

Microsoft Visual Studio is available in both Windows and Mac versions, for use on any form of machine. Microsoft Studio 2017 is a full-featured IDE for efficient app creation, while Visual Studio for Mac is a mobile-first, cloud-first IDE made for the Mac. Both have robust features that make it simple to build, debug, collaborate, and extend.

Figure 15 - Visual Studio

MonoDevelop is a cross-platform IDE that supports C#, F#, and other programming languages with ease of integration and the ability to quickly create desktop and mobile apps. MonoDevelop was commonly used for the smooth incorporation of code into Unity Editor, starting as the first IDE ever supported by Unity. Unity used to support MonoDevelop as an in-built package until Unity 2018, but it was then discontinued and replaced by Microsoft Visual Studio as the default IDE. MonoDevelop can still be downloaded for free from their official website.

Figure 16 - MonoDevelop

Microsoft Visual Studio Code is a free and open source code editor. TypeScript, JavaScript, and Node.js are all supported by this free text editor. Smart completions are based on variable types, critical modules, and function descriptions, and it is autocompleted with IntelliSense features.

Figure 17 - Visual Studio Code

Adobe Brackets, an open-source editor, appears to be a well-rounded piece of software. It does not accept as many languages as some of the others when it comes to syntax highlighting. It also supports CSS pre-processors like Less and Sass because of its emphasis on front end technologies. Brackets is not the fastest or most reliable on all of the usual metrics, but it does have a few unique features worth investigating. Most of the other editors require us to edit configuration files, while it is mostly configurable via its menus.

Figure 18 - Adobe Brackets

Atom is open source and developed by GitHub. It was heavily influenced by the modern style of editor popularized by Sublime Text during its initial development. However, there are a few main distinctions: Atom is a free and open source text editor that comes with built-in support for Git and GitHub. Atom has had performance and stability issues in the past, but these have greatly improved as the technology has matured.

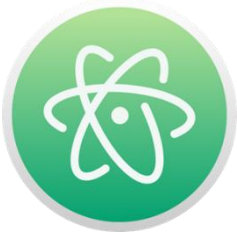


Figure 19 - Atom

2.5 - Conclusion

An unanimous agreement on an offline arcade-style 3D car racing video game called Hyperdrive was reached, after watching several YouTube videos on how to develop a video game, where the player must race against AI-controlled automobiles. Unity was chosen as game engine for a simple start as beginners, with the stable version being **Unity 2020**, alongside **Microsoft Visual Studio,** the default Unity IDE. **Blender** is used to model the cars, tracks, and terrain, and then imported into Unity.

3 - Analysis

3.1 - Proposed System

The game is an offline arcade-style 3D car racing video game where the player can race in different levels using different cars and playing different modes.

The game consists of 5 different levels with different maps namely Rural, Arctic, Desert, Urban, Moon. There are 5 different cars that the player can choose from and they can finally select the mode in which they want to play.

The player can choose to play the career mode whereby most cars, tracks and game modes are locked, and they have to be unlocked. To unlock them, the player has to race against AI-driven cars and finish first. The player is rewarded with stars for finishing a level which they can use to unlock the items mentioned above. The goal of the player in this mode is to unlock all the different levels, cars, and game modes.

The player can also choose the "Quick Race" option whereby they can choose any car, track, or game mode without having unlocked them in the career mode. In this situation, the player is also able to choose the difficulty of the game, the number of laps they want to perform and the number of opponents they want to race against.

The game is called Hyperdrive.

3.2 - Functional Requirements

3.2.1 - Gameplay

- The game shall have 5 levels.
- The game shall have 5 different cars.
- The game shall have 5 game modes.
- The system shall allow the player to use the arrow keys (Player A) or the WASD keys (Player B) to drive the cars.
- The system shall automatically save the player's progress.
- The system shall allow the player to choose the difficulty level.

3.2.2 - Main Menu Screen

- The system shall display the main menu screen when the game is launched.
- The system shall allow the player to choose between Play and Quick Race.

3.2.3 - Options Button

- The system shall be able to adjust volume level.
- The system all allow the player to choose a song.
- The system shall allow the player to reset the game progress.

3.2.4 - Help Button

- The system shall explain to the player the game controls.
- The system shall explain the different game modes.

3.2.5 - Credits Button

- The system shall display the game developers' names.

3.2.6 - Exit Button

- The system shall allow the player to quit the game.

3.2.7 - Play Button

- The system shall allow the player to play the career mode.

3.2.8 - Quick Race Button

- The system shall allow the player to play the 1-player or 2-player quick race.

3.2.9 - Level Selection Screen

- The system shall allow the player to choose the desired level.

3.2.10 - Race Settings Screen

- The system shall allow the player to choose the race mode.
- The system shall allow the player to choose difficulty.
- The system shall allow the player to choose number of laps.
- The system shall allow the player to choose number of cars.

3.2.11 - Car Selection Screen

- The system shall allow the player to choose a car.

3.2.12 - Race Screen

- The system shall display the number of laps.
- The system shall display the player's rank in real-time.
- The system shall display the car's speed in real-time.
- The system shall display the lap time in real-time.
- The system shall display the last lap time.
- The system shall display the best lap time.

3.2.13 - Pause Button

- The system shall allow the player to pause the game.

3.2.14 - Pause Menu Screen

- The system shall allow the player to change volume level.
- The system shall display the race level, map, mode, and difficulty.
- The system shall allow the player to change songs.
- The system shall allow the player to choose either "Quit", "Restart" or "Resume".

3.2.15 - Restart Button

- The system shall ask the player whether they want to restart the race or not.

3.2.16 - Quit Button

- The system shall ask the player whether they want to quit the race or not.
- The system shall return the player to the main menu screen if they choose to.

3.2.17 - Exit Button

- The system shall ask the player whether they want to exit the game or not.

3.2.18 - Reset Game Button

- The system shall ask the player whether they want to reset the game progress or not.

3.2.19 - Race Over Screen

- The system shall display the number of stars.
- The system shall display the rank of the player.
- The system shall display the race level, map, mode, and difficulty.

- The system shall display the Restart and Next button.

3.3 - Non-Functional Requirements

3.3.1 - Performance

- The system shall respond within 1 second of when player clicks a button.
- The system shall load the level within 5 seconds.

3.3.2 - Storage

- The system shall require at most 500 MB of available disk space.
- The system shall occupy 1 GB of RAM at most while running.

3.3.3 - Maintainability

- The system shall load the game from where the player left off keeping all of their unlocked items.
- The system shall allow code written for the game to be maintainable.

3.3.4 - Reliability

- The system shall have a defect rate nor exceeding 1 failure per 100 hours of operation.
- The systems shall run without any stutter or buffer.
- The system shall be reliable at least 95% of the time.

3.3.5 - Implementation

- The system shall be implemented using Unity.
- The system shall be able to run on Windows.
- The system shall use C# as the programming language to implement the game.
- The system shall be compiled and built using Unity.

3.3.6 - Usability

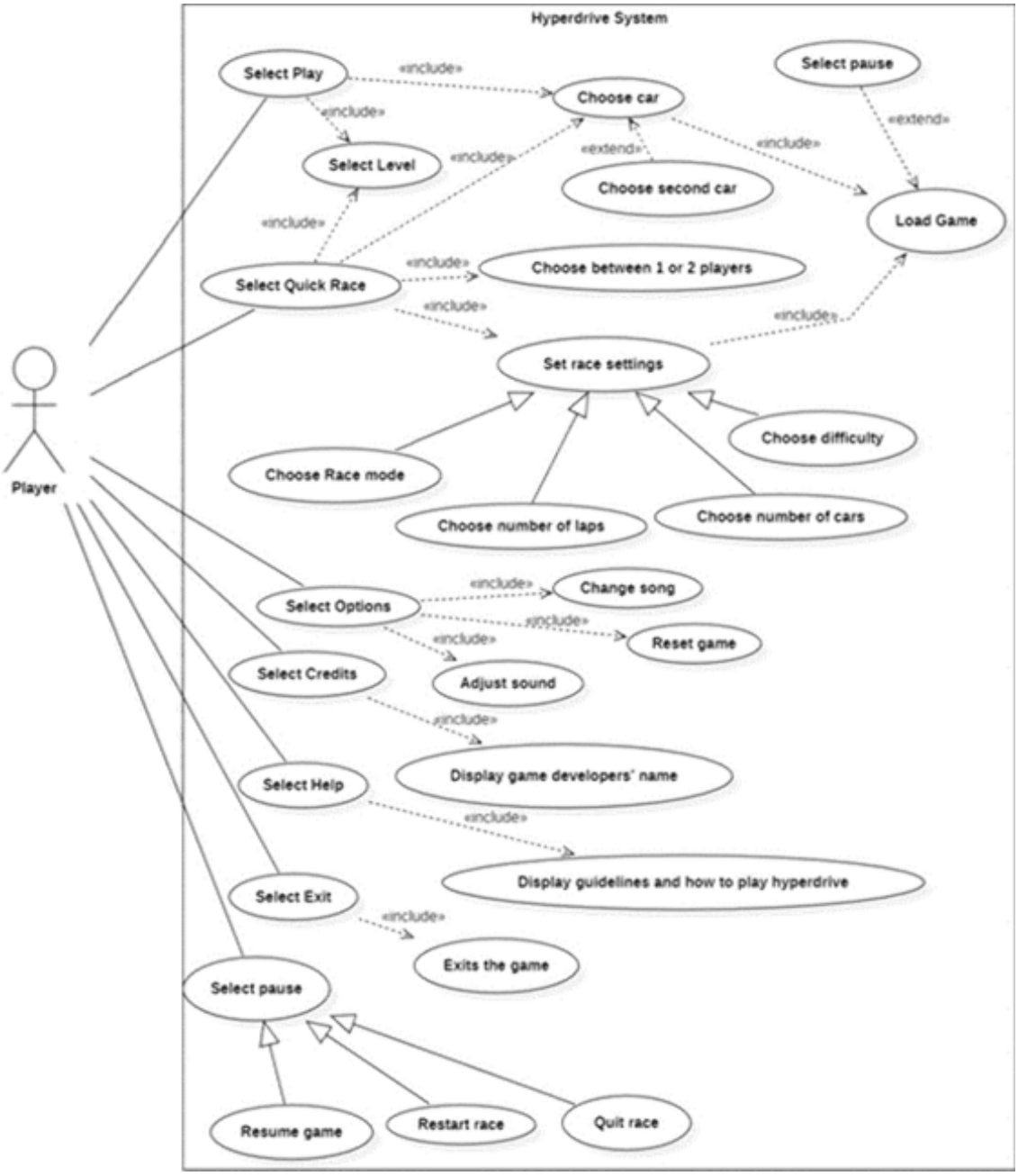
- The system shall have an average frame rate of 30 to 60 frames per second.
- The system shall have an interactive interface to facilitate use.

3.3.7 - Safety

- The system shall not contain any adult materials.

3.4 - Use Case Diagram

Use Case Name	Select Quick Race
Participating Actors	User
Pre-Condition	The system is working.
Basic Flow of Events	1. The system displays the Menu Screen to the user. 2. The user selects the Quick Race button. 3. The system prompts the user to select between 1 or 2 players. 4. The user selects the number of players. 5. The system validates the number of players. 6. The system displays the level selection screen and asks the user to choose a level. 7. The user selects the level they want to play. 8. The system displays the Race settings screen and ask to user to modify the race settings. 9. The user modifies the race settings accordingly. 10. The system prompts the car selection screen and asks the suer to choose a car. 11. The user selects his car. 12. The game is loaded. 13. The use case ends.
Alternative Flows	A. The user selects 2-players. 1. If in step 7 of the basic flow of events, the user selects 2 players, then, 2. The System loads 2 consecutive car selection screens. 3. First user chooses his/her car. 4. Second user chooses his/her car. 5. The use case resumes at step 12. B. The user selects Back button. 6. If in step 7 of the basic flow of events, the user selects Back button, then, 7. The use case resumes at step 1.
Post-Condition	The game is loaded, and the user can play the game if they select the 1 or 2 player mode, modify the race settings, and selects his/her car.


Hyperdrive System
Player
Select Play
Select Level
Select Quick Race
Choose car
Choose second car
Choose between 1 or 2 players
Set race settings
Choose Race mode
Choose number of laps
Choose difficulty
Choose number of cars
Load Game
Select pause
Select Options
Change song
Reset game
Adjust sound
Select Credits
Display game developers' name
Select Help
Display guidelines and how to play hyperdrive
Select Exit
Exits the game
Select pause
Resume game
Restart race
Quit race
«include»
«extend»

3.5 - Tools Chosen

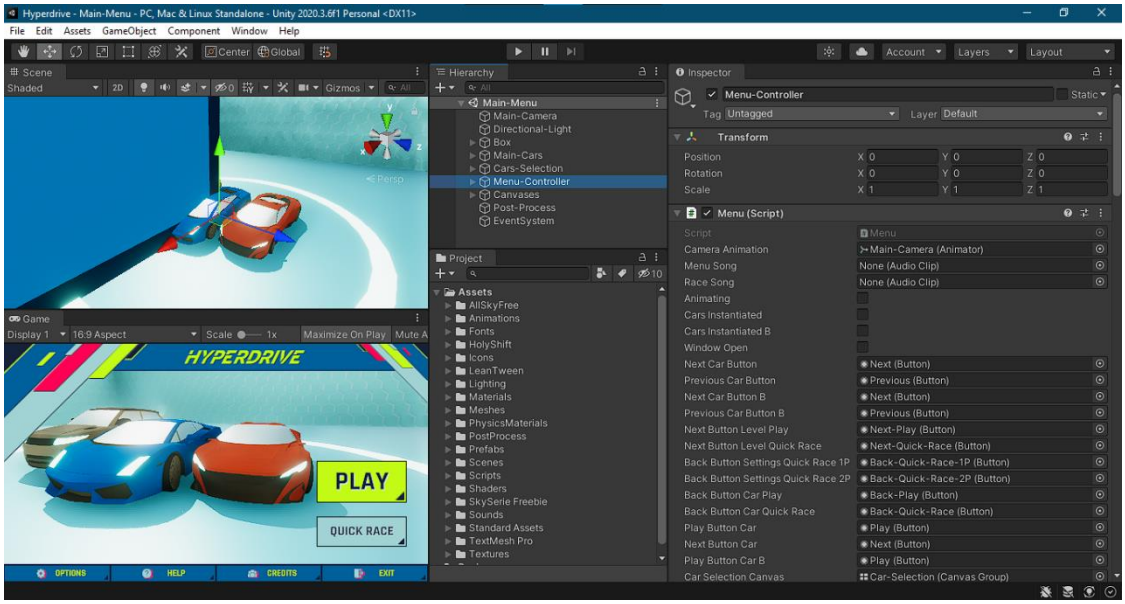
The game engine chosen is **Unity 2020**. It can be used to build both 3D and 2D games. Unity was chosen as the game engine because it is very straightforward to use for inexperienced developers and since it has been utilised by many independent game creators. Furthermore, various tutorials on how to use the Unity engine are available online, making the development process easier.

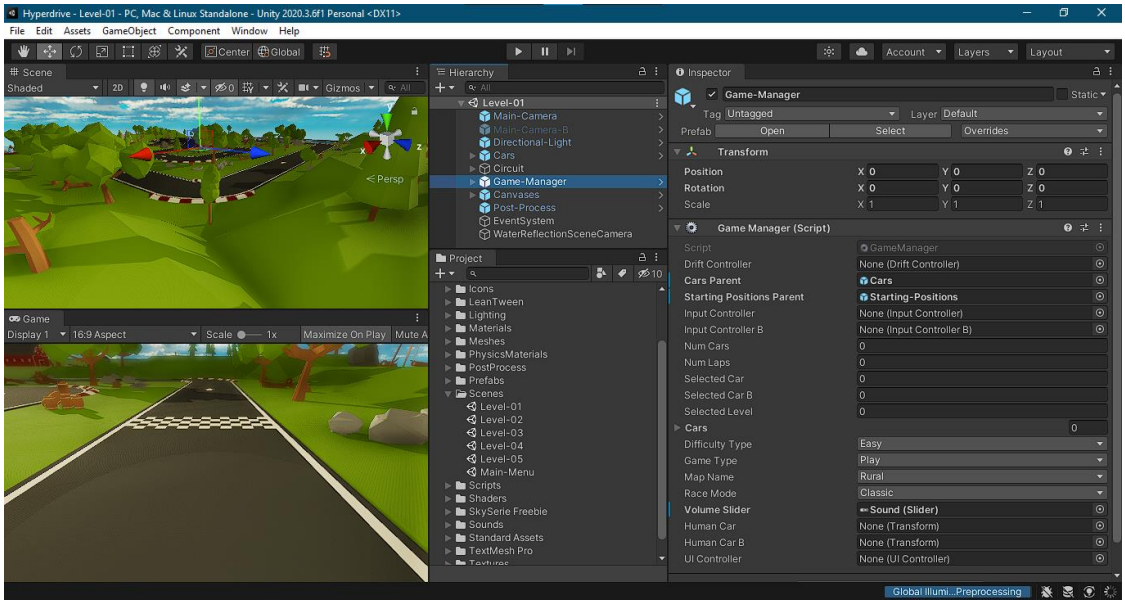
C# is the programming language that is used to create the game. C# is really close to C and C++, and because we are currently studying C++ with Mr. Somveer Kishnah, we are getting used to it quickly. Additionally, when dealing with Unity, C# is the most commonly used programming language. The code editor used is **Microsoft Visual Studio**.

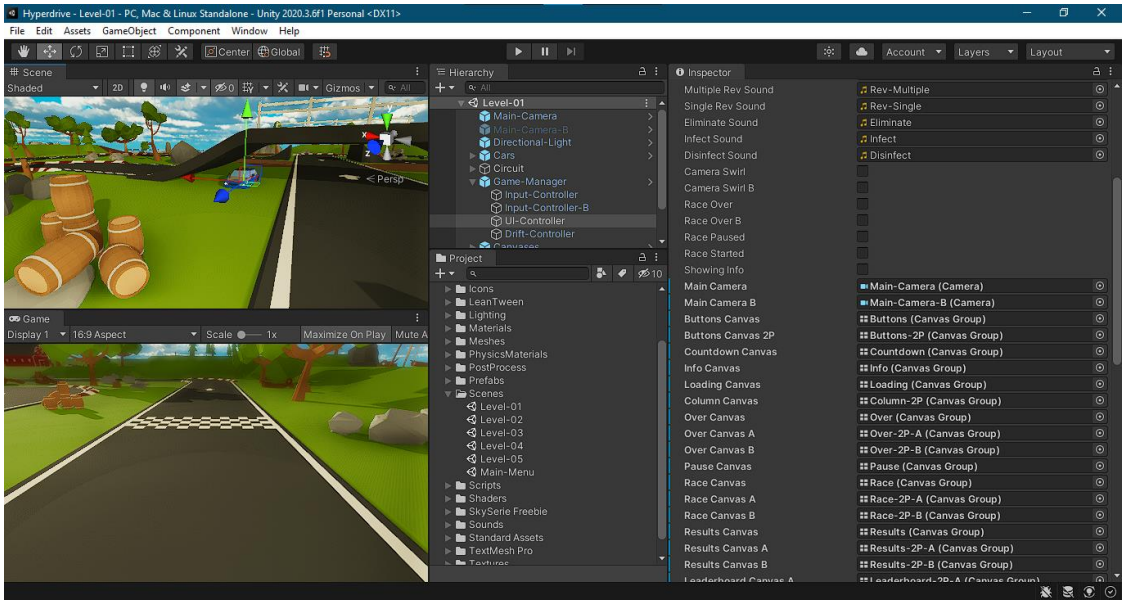
Blender is used to build the game's various properties, such as the vehicles, tracks, and various props including trees and rocks. Blender can be difficult to learn at first, but with the help of numerous online tutorials, we can quickly adapt and build the assets needed.

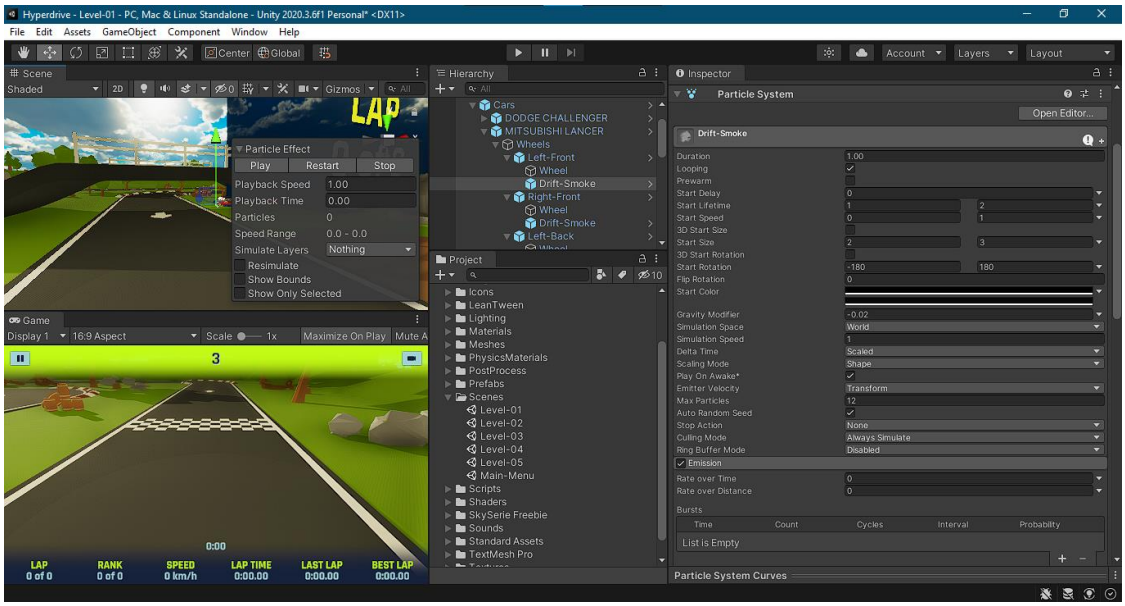
4 - Design

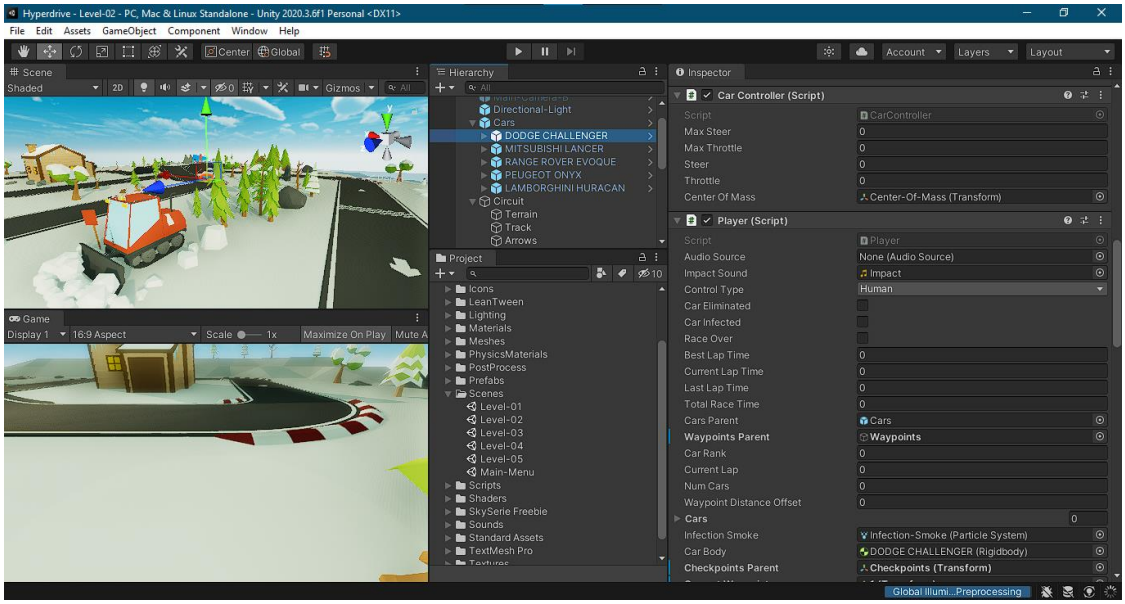
4.1 - Architectural Design

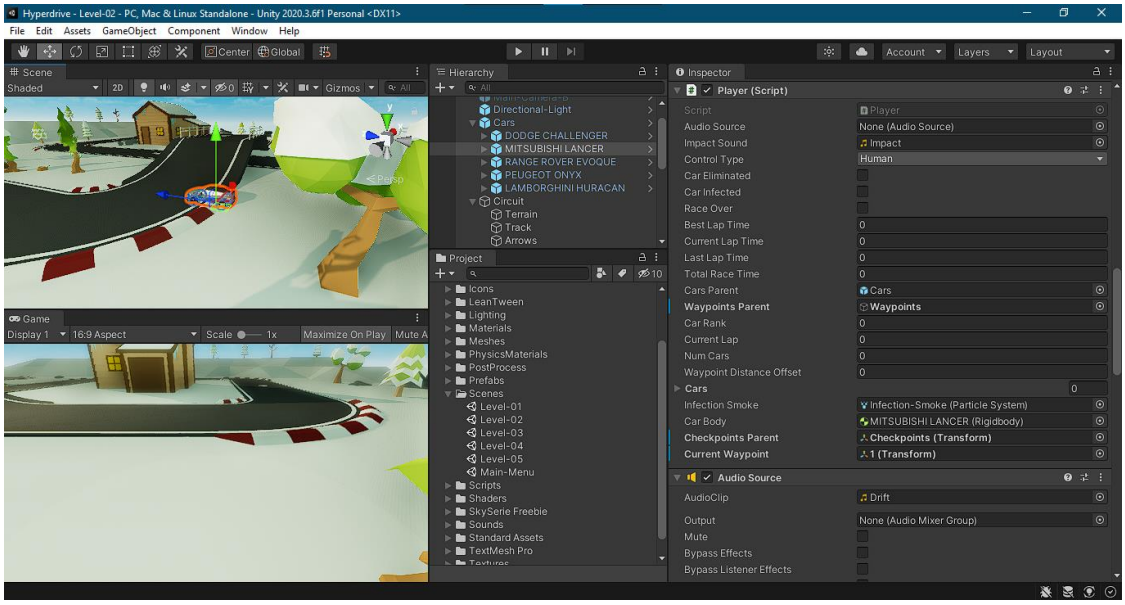


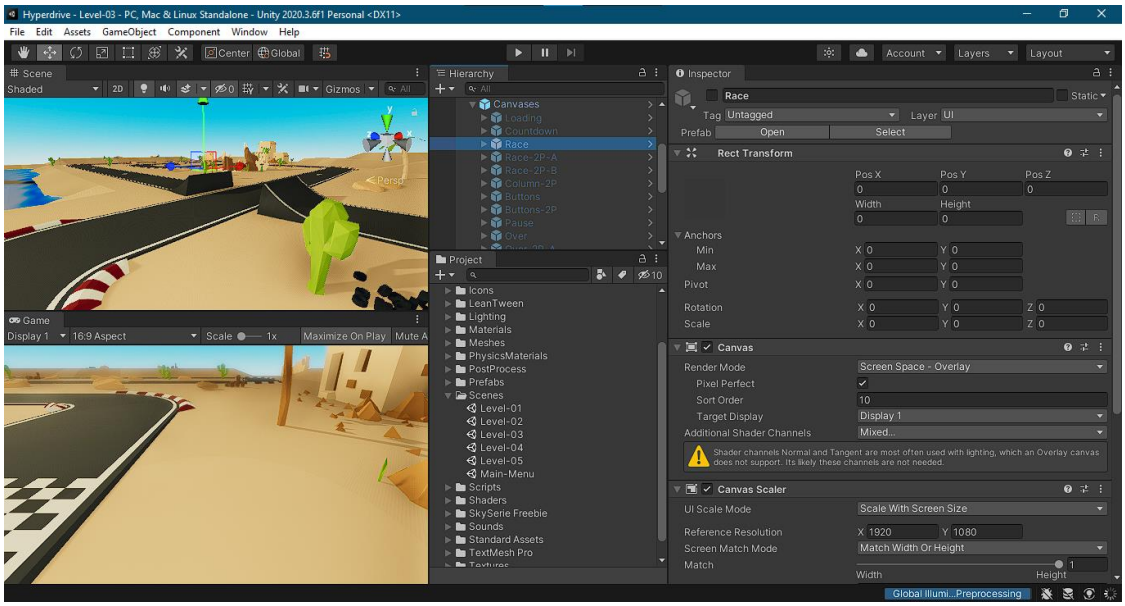


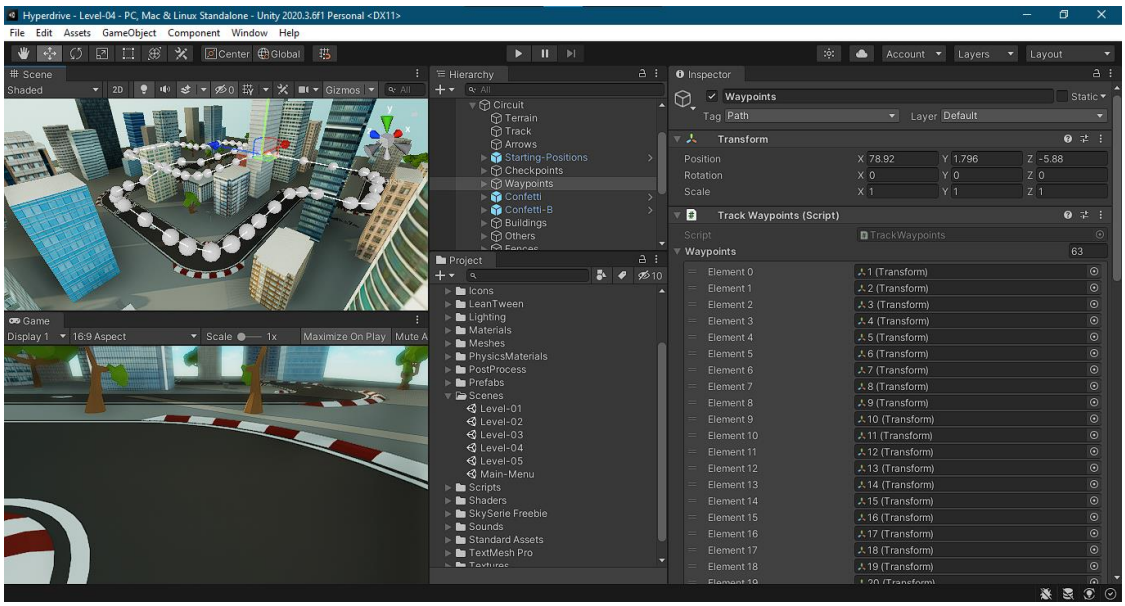

Hyperdrive - Level-01 - PC, Mac & Linux Standalone - Unity 2020.3.6f1 Personal <DX11>
File Edit Assets GameObject Component Window Help
Scene
Hierarchy
Inspector
Project
Game
Level-01
Main-Camera
Directional-Light
Cars
Circuit
Game-Manager
Input-Controller
UI-Controller
Drift-Controller
Multiple Rev Sound
Single Rev Sound
Eliminate Sound
Infect Sound
Disinfect Sound
Main Camera
Main Camera B
Buttons Canvas
Buttons Canvas 2P
Countdown Canvas
Info Canvas
Loading Canvas
Column Canvas
Over Canvas
Over Canvas A
Over Canvas B
Pause Canvas
Race Canvas
Race Canvas A
Race Canvas B
Results Canvas
Results Canvas A
Results Canvas B


Hyperdrive - Level-01 - PC, Mac & Linux Standalone - Unity 2020.3.6f1 Personal* <DX11>
File Edit Assets GameObject Component Window Help
Particle Effect
Play
Restart
Stop
Playback Speed
Playback Time
Particles
Speed Range
Simulate Layers
Resimulate
Show Bounds
Show Only Selected
LAP
3
0:00
LAP
0 of 0
RANK
0 of 0
SPEED
0 km/h
LAP TIME
0:00.00
LAST LAP
0:00.00
BEST LAP
0:00.00
Particle System
Open Editor...
Drift-Smoke
Particle System Curves


Hyperdrive - Level-02 - PC, Mac & Linux Standalone - Unity 2020.3.6f1 Personal <DX11>
File Edit Assets GameObject Component Window Help
Car Controller (Script)
Max Steer
Max Throttle
Steer
Throttle
Center Of Mass
Player (Script)
Audio Source
Impact Sound
Control Type
Car Eliminated
Car Infected
Race Over
Best Lap Time
Current Lap Time
Last Lap Time
Total Race Time
Cars Parent
Waypoints Parent
Car Rank
Current Lap
Num Cars
Waypoint Distance Offset
Infection Smoke
Car Body
Checkpoints Parent


Hyperdrive - Level-02 - PC, Mac & Linux Standalone - Unity 2020.3.6f1 Personal <DX11>
File Edit Assets GameObject Component Window Help
Hierarchy
Inspector
Player (Script)
Audio Source
Project
Game


Hyperdrive - Level-03 - PC, Mac & Linux Standalone - Unity 2020.3.6f1 Personal <DX11>
File Edit Assets GameObject Component Window Help
Hierarchy
Inspector
Race
Rect Transform
Canvas
Canvas Scaler
Project
Game


Hyperdrive - Level-04 - PC, Mac & Linux Standalone - Unity 2020.3.6f1 Personal <DX11>
File Edit Assets GameObject Component Window Help
Hierarchy
Inspector
Waypoints
Transform
Track Waypoints (Script)
Project
Game

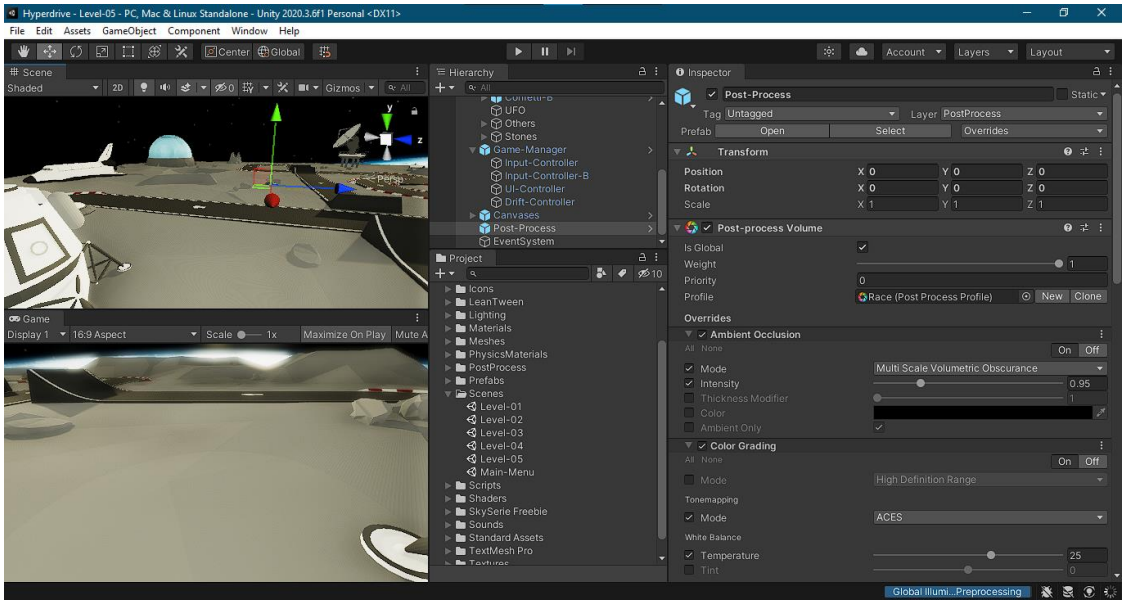
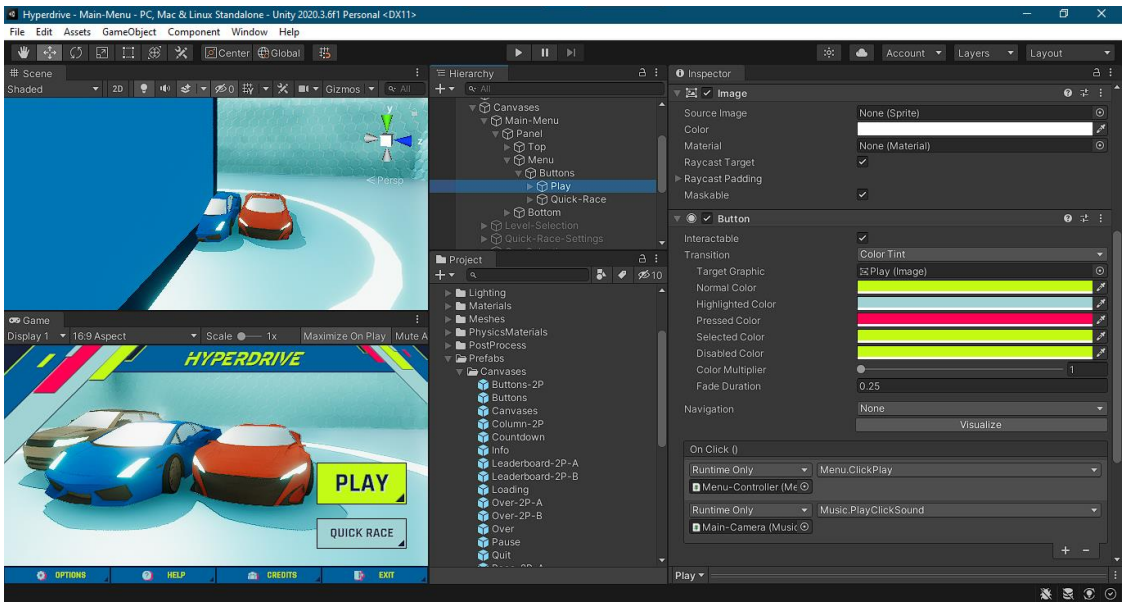
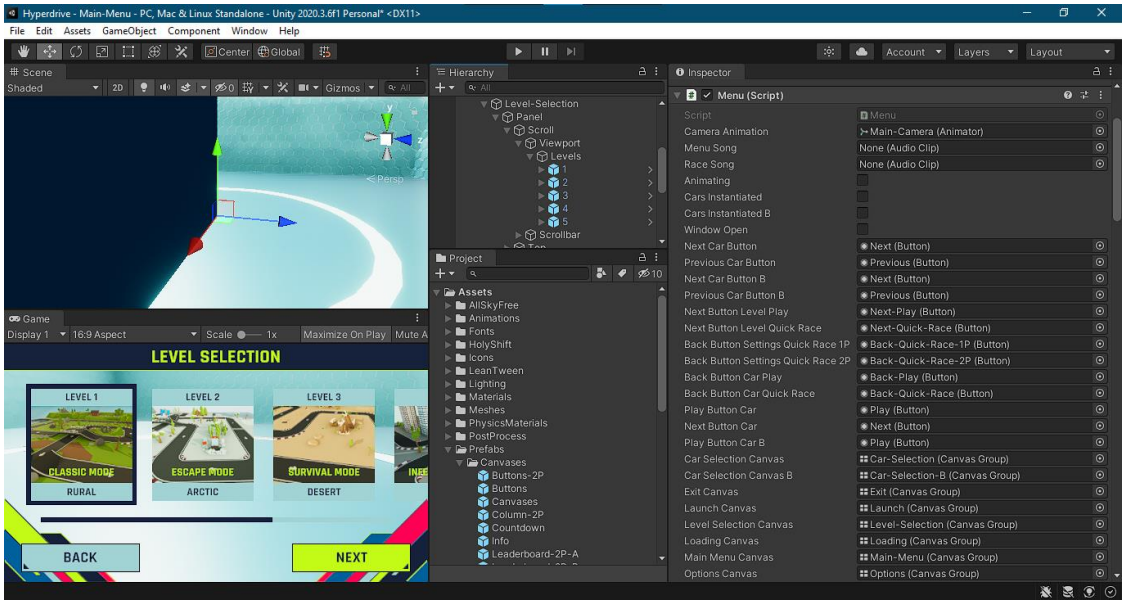
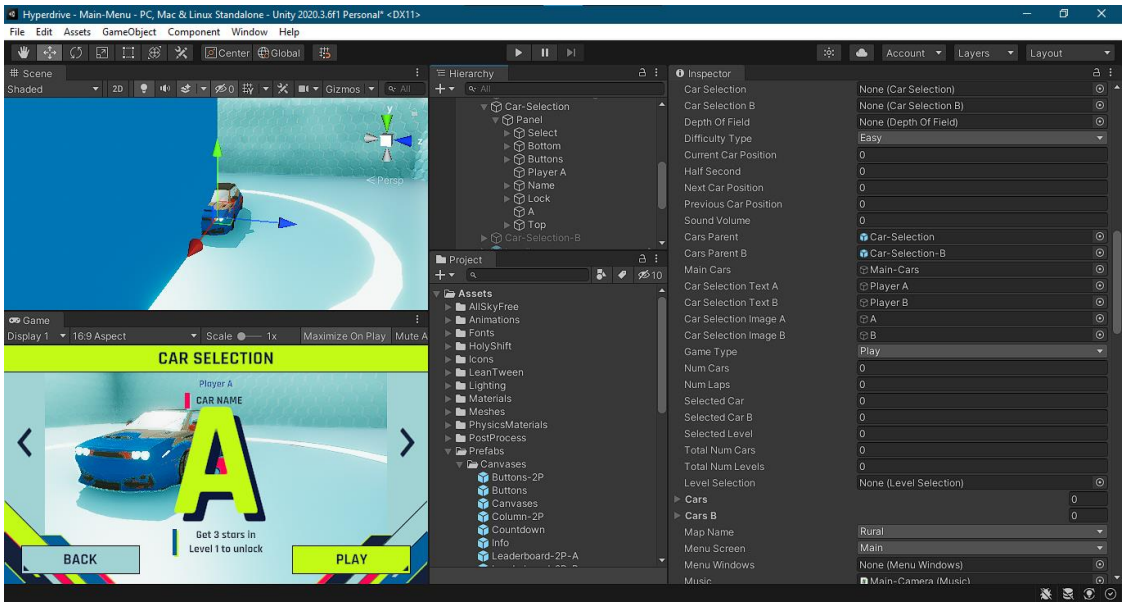


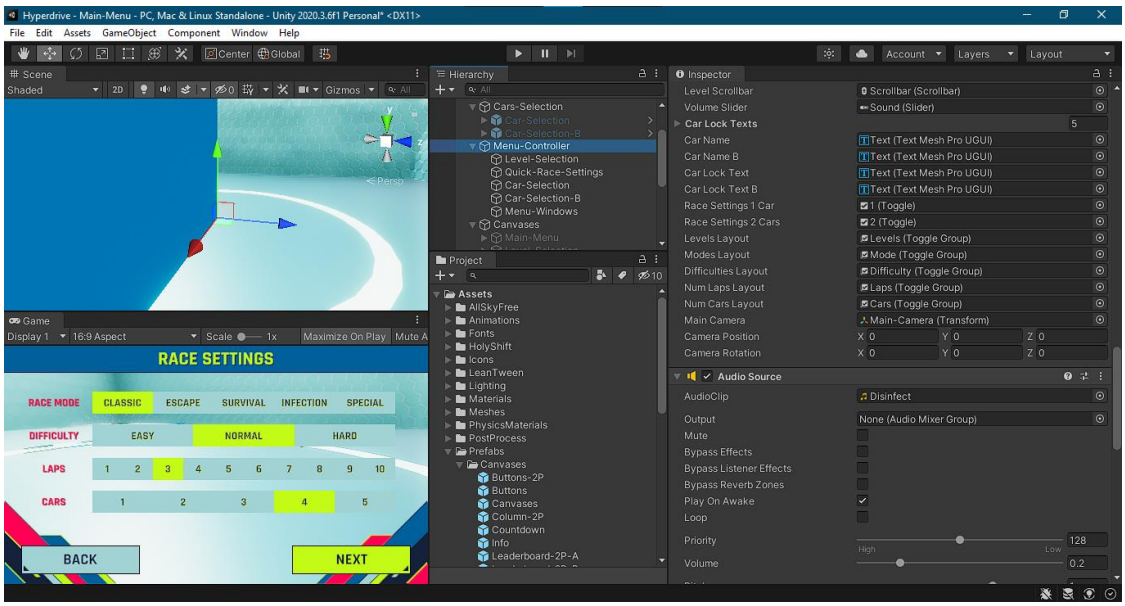
Figure 20 - Hyperdrive Project in Unity 2020

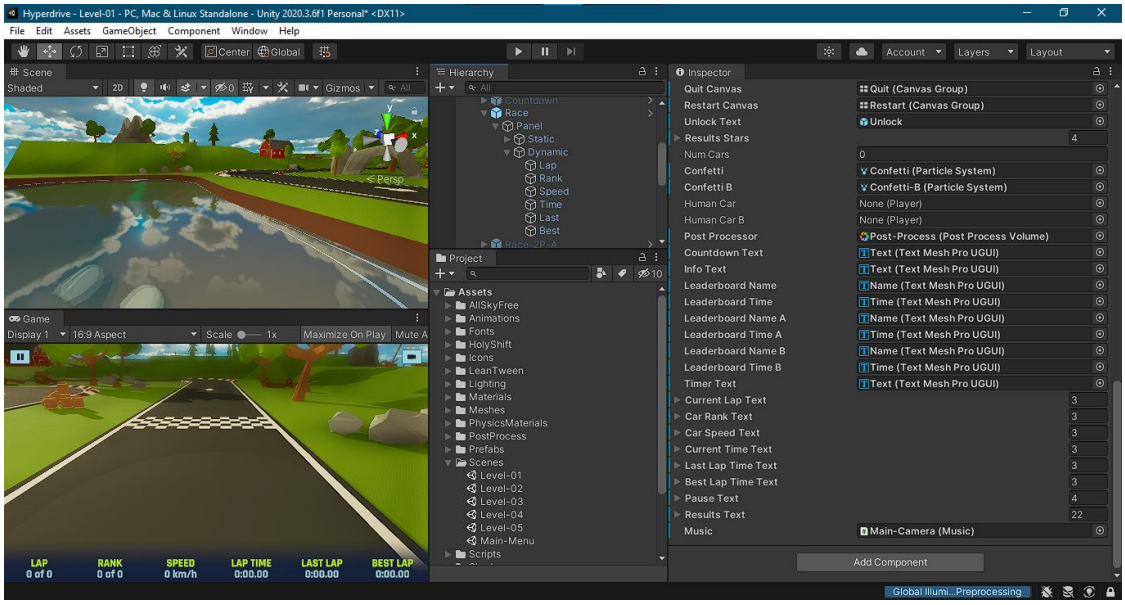
4.2 - UI DESIGN

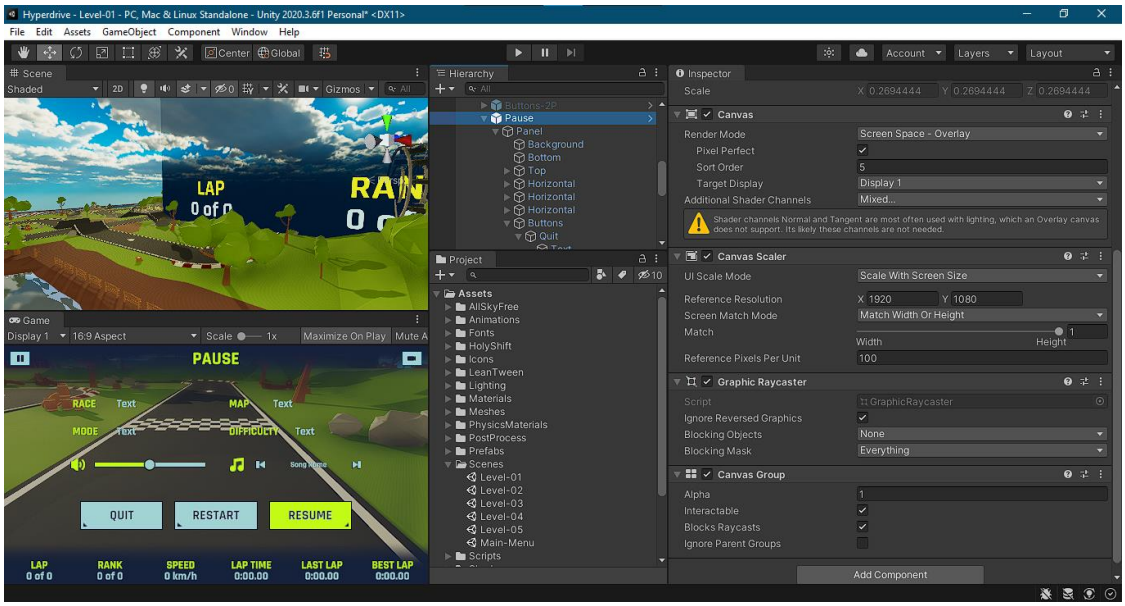


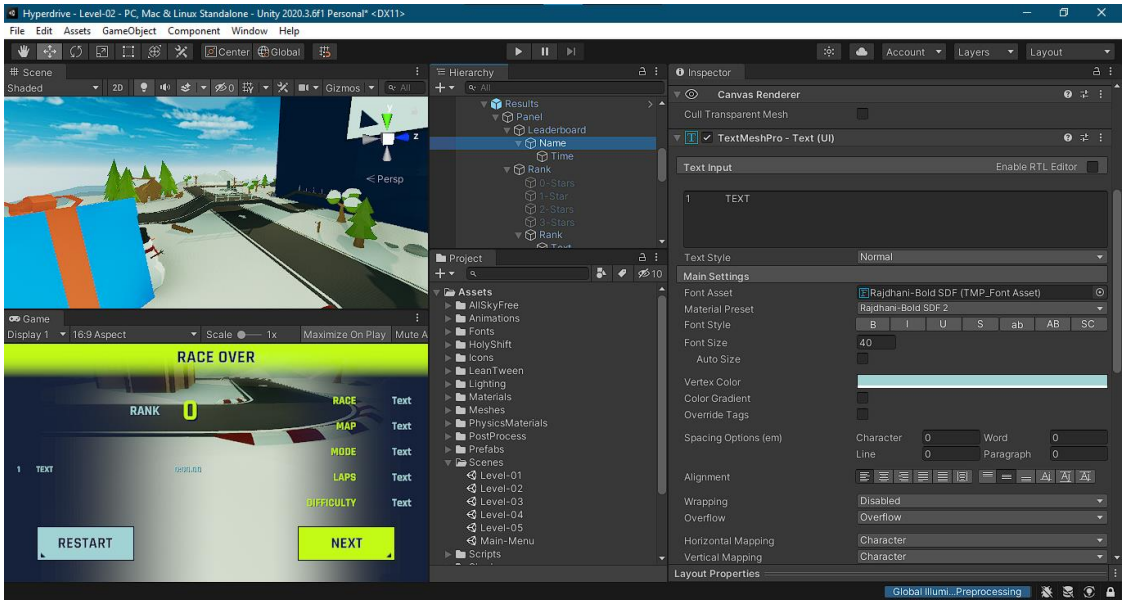

Hyperdrive - Main-Menu - PC, Mac & Linux Standalone - Unity 2020.3.6f1 Personal* <DX11>
File Edit Assets GameObject Component Window Help
Scene
Game
Hierarchy
Project
Inspector
LEVEL SELECTION
LEVEL 1
LEVEL 2
LEVEL 3
CLASSIC MODE
RURAL
ESCAPE MODE
ARCTIC
SURVIVAL MODE
DESERT
BACK
NEXT
Menu (Script)

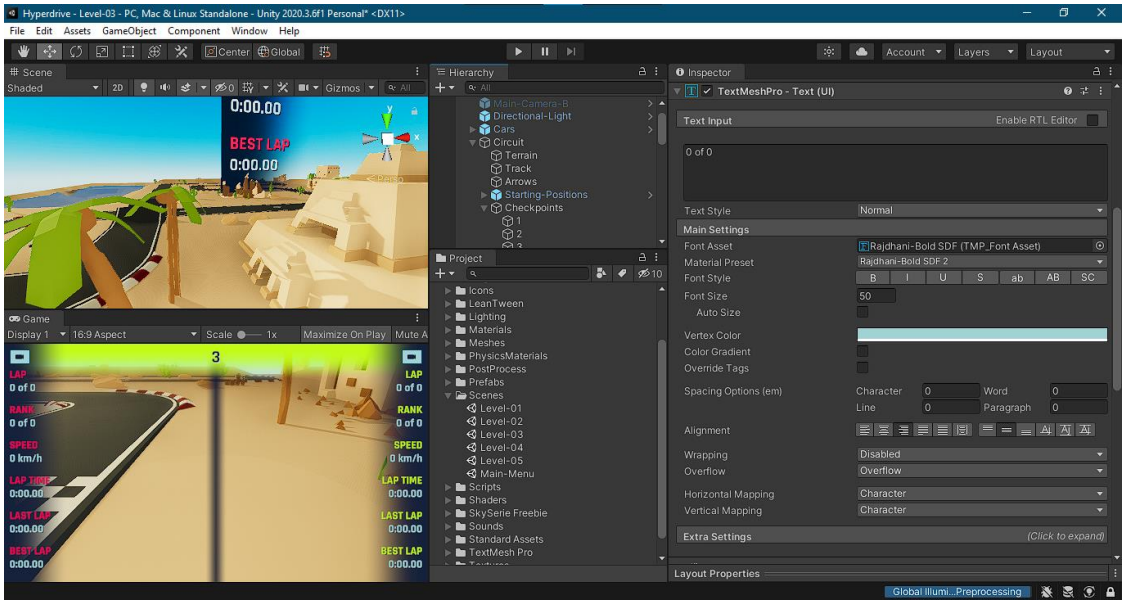

Hyperdrive - Main-Menu - PC, Mac & Linux Standalone - Unity 2020.3.6f1 Personal* <DX11>
File Edit Assets GameObject Component Window Help
Scene
Game
Hierarchy
Project
Inspector
CAR SELECTION
Player A
CAR NAME
A
Get 3 stars in
Level 1 to unlock
BACK
PLAY

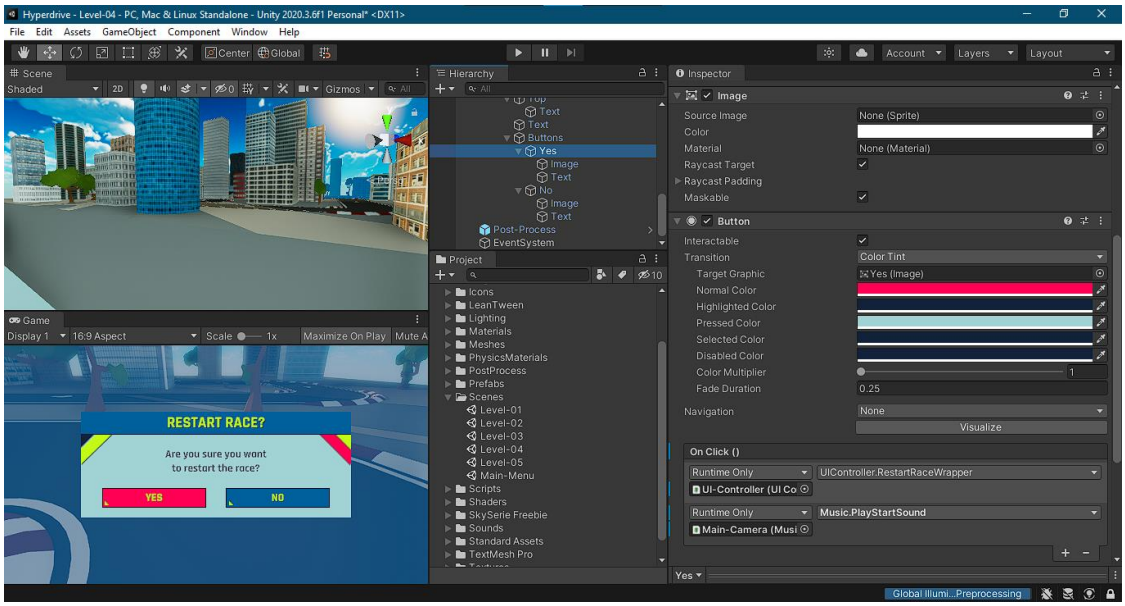

Hyperdrive - Main-Menu - PC, Mac & Linux Standalone - Unity 2020.3.6f1 Personal* <DX11>
File Edit Assets GameObject Component Window Help
Scene
Game
Hierarchy
Project
Inspector
RACE SETTINGS
RACE MODE
CLASSIC
ESCAPE
SURVIVAL
INFECTION
SPECIAL
DIFFICULTY
EASY
NORMAL
HARD
LAPS
CARS
BACK
NEXT
Audio Source


Hyperdrive - Level-01 - PC, Mac & Linux Standalone - Unity 2020.3.6f1 Personal* <DX11>
File Edit Assets GameObject Component Window Help
Scene
Game
Hierarchy
Project
Inspector
Quit Canvas
Restart Canvas
Unlock Text
Results Stars
Num Cars
Confetti
Confetti B
Human Car
Human Car B
Post Processor
Countdown Text
Info Text
Leaderboard Name
Leaderboard Time
Leaderboard Name A
Leaderboard Time A
Leaderboard Name B
Leaderboard Time B
Timer Text
Current Lap Text
Car Rank Text
Car Speed Text
Current Time Text
Last Lap Time Text
Best Lap Time Text
Pause Text
Results Text
Music
Add Component
LAP
RANK
SPEED
LAP TIME
LAST LAP
BEST LAP


Hyperdrive - Level-01 - PC, Mac & Linux Standalone - Unity 2020.3.6f1 Personal* <DX11>
File Edit Assets GameObject Component Window Help
Canvas
Canvas Scaler
Graphic Raycaster
Canvas Group
Add Component
PAUSE
QUIT
RESTART
RESUME
LAP
RANK
SPEED
LAP TIME
LAST LAP
BEST LAP


Hyperdrive - Level-02 - PC, Mac & Linux Standalone - Unity 2020.3.6f1 Personal* <DX11>
File Edit Assets GameObject Component Window Help
Canvas Renderer
TextMeshPro - Text (UI)
RACE OVER
RANK
RESTART
NEXT


Hyperdrive - Level-03 - PC, Mac & Linux Standalone - Unity 2020.3.6f1 Personal* <DX11>
File Edit Assets GameObject Component Window Help
Scene
Hierarchy
Inspector
Project
Game
TextMeshPro - Text (UI)
Text Input
Main Settings
Font Asset
Material Preset
Font Style
Font Size
Vertex Color
Color Gradient
Override Tags
Spacing Options (em)
Alignment
Wrapping
Overflow
Horizontal Mapping
Vertical Mapping
Extra Settings
Layout Properties
0:00.00
BEST LAP
0:00.00
LAP
RANK
SPEED
LAP TIME
LAST LAP
BEST LAP


Hyperdrive - Level-04 - PC, Mac & Linux Standalone - Unity 2020.3.6f1 Personal* <DX11>
File Edit Assets GameObject Component Window Help
Scene
Hierarchy
Inspector
Project
Game
Image
Button
On Click ()
RESTART RACE?
Are you sure you want
to restart the race?
YES
NO

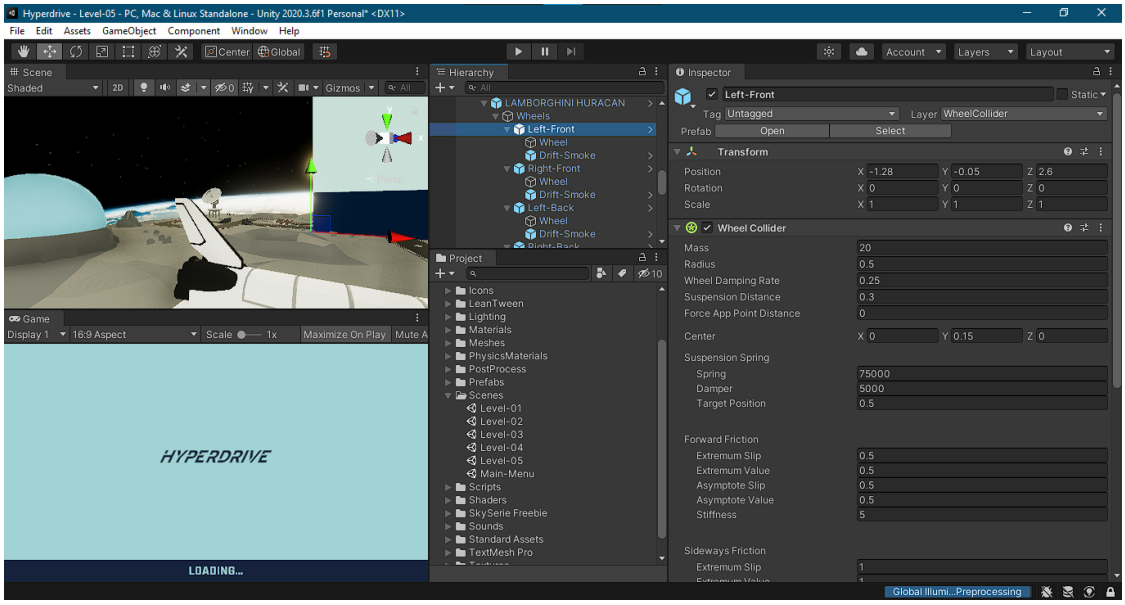
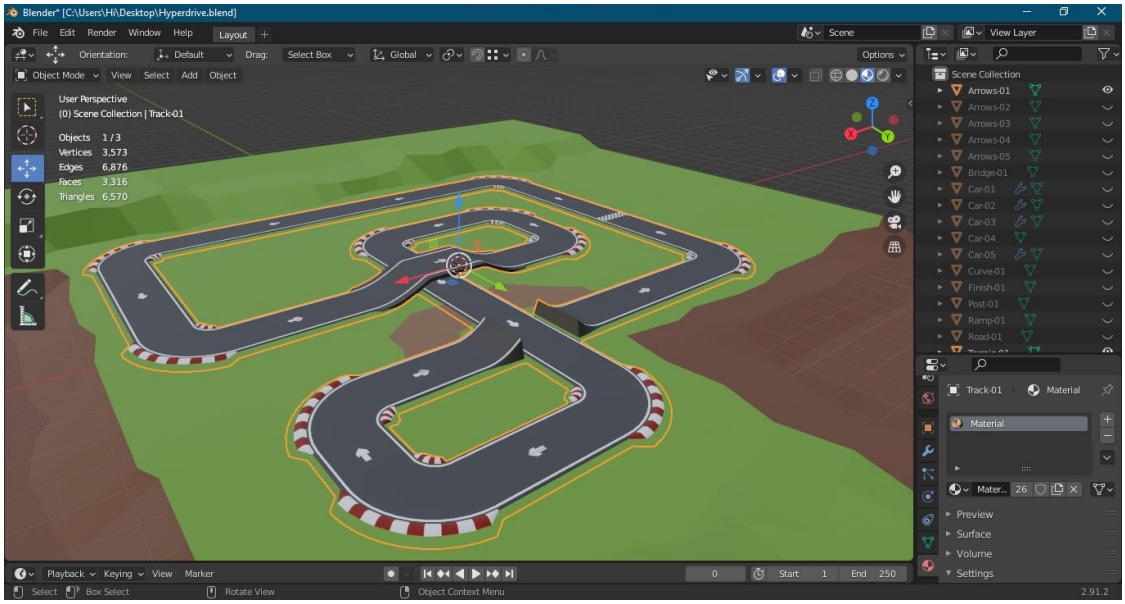
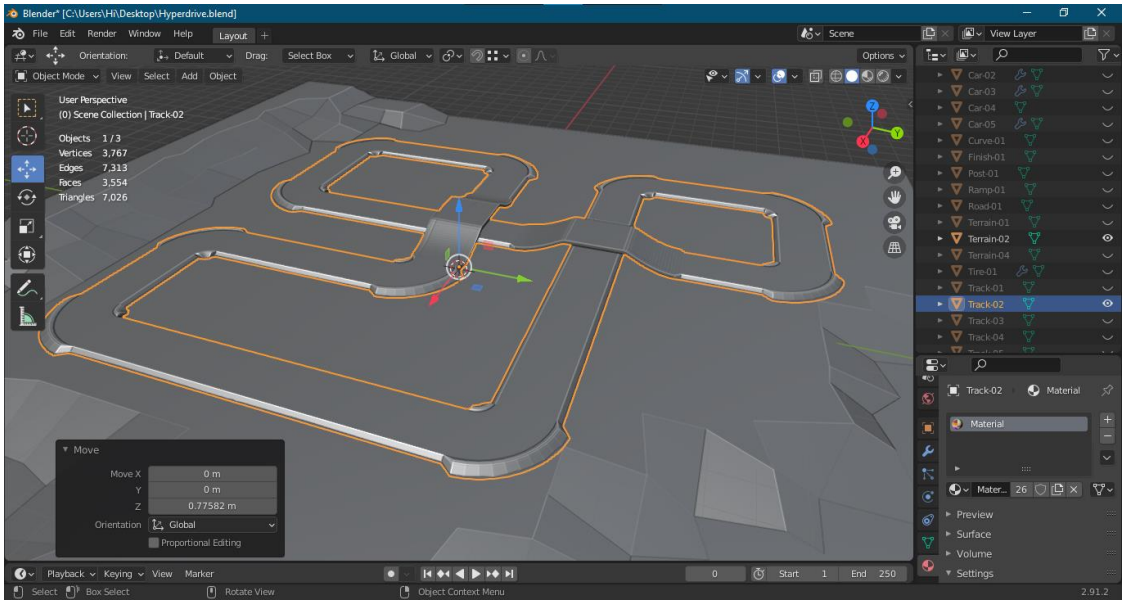
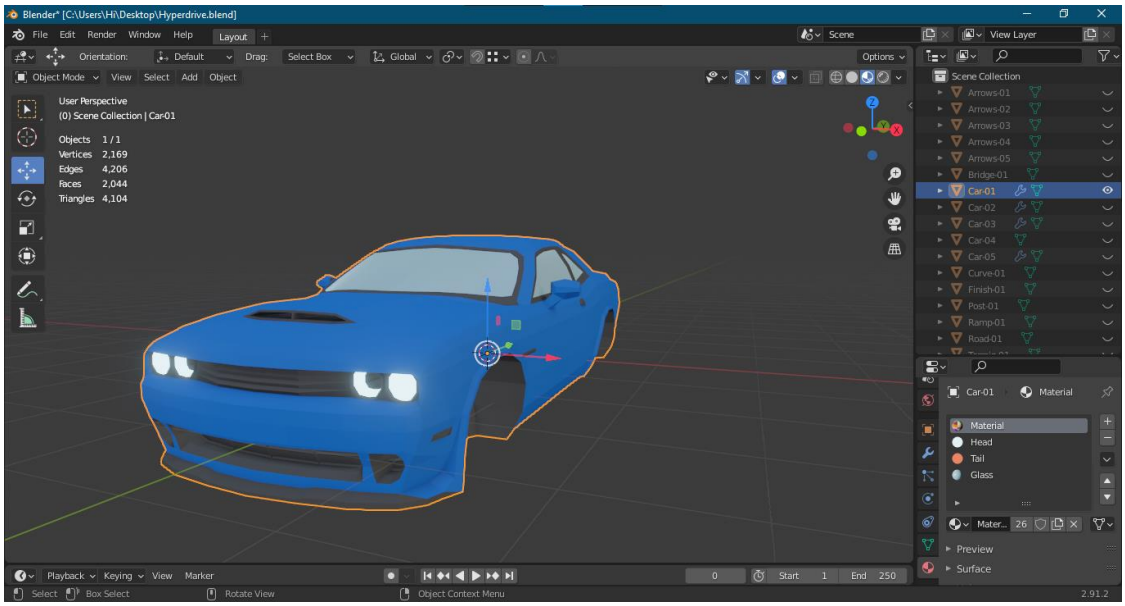


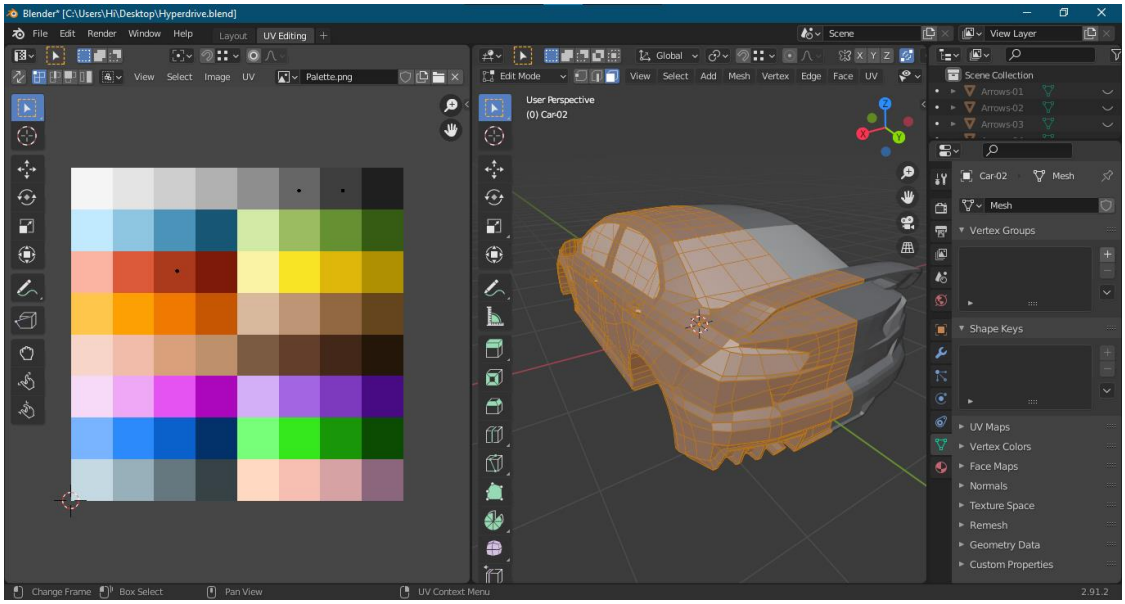
Figure 21 - Hyperdrive UI Design

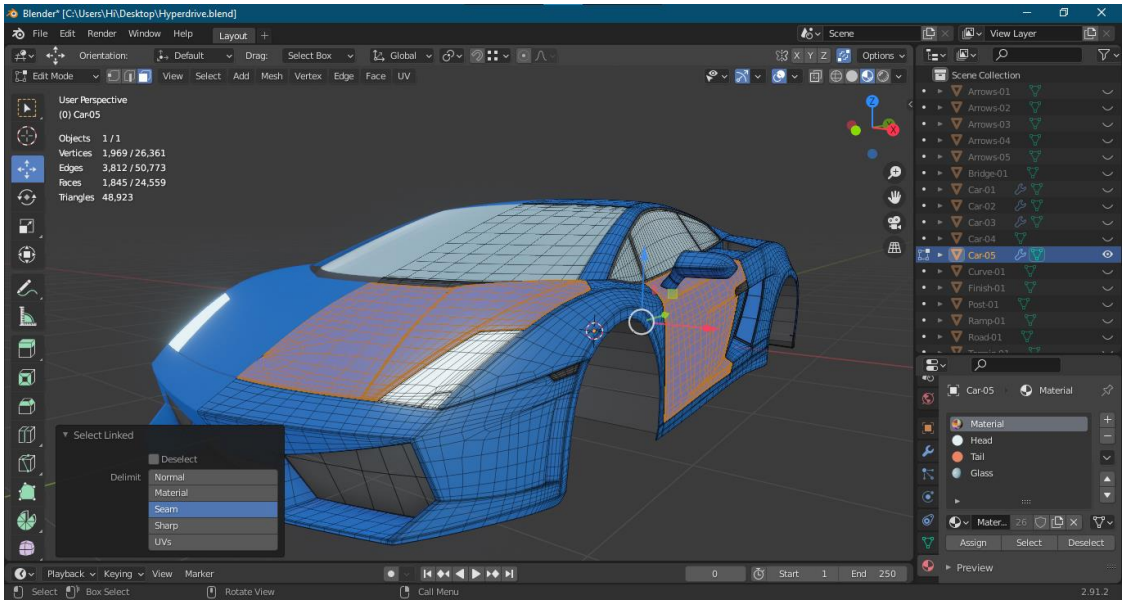
4.3 - Models and Meshes

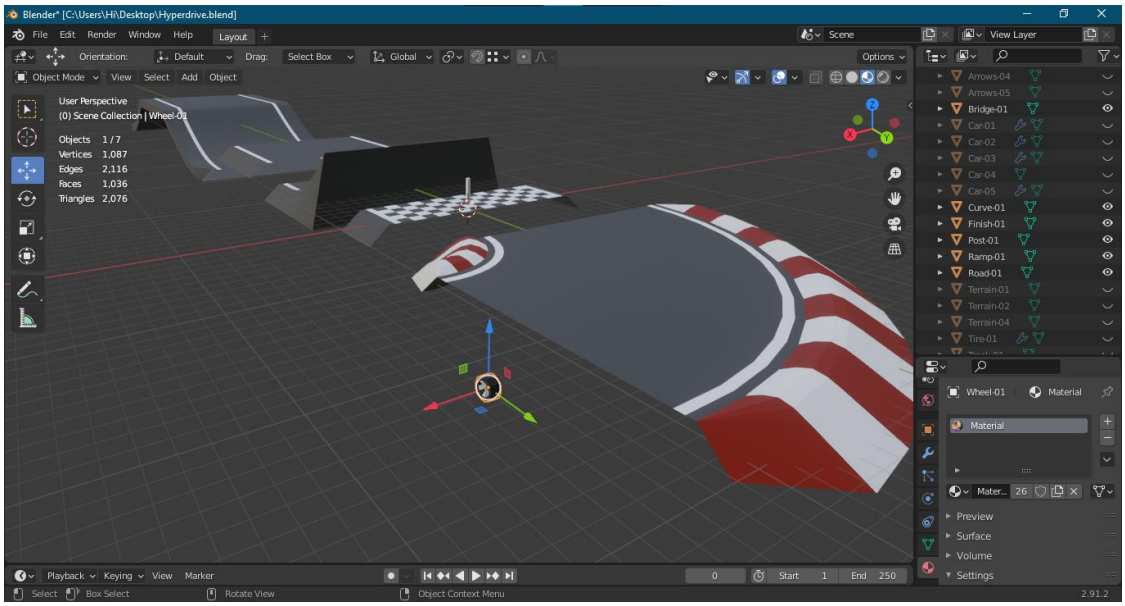


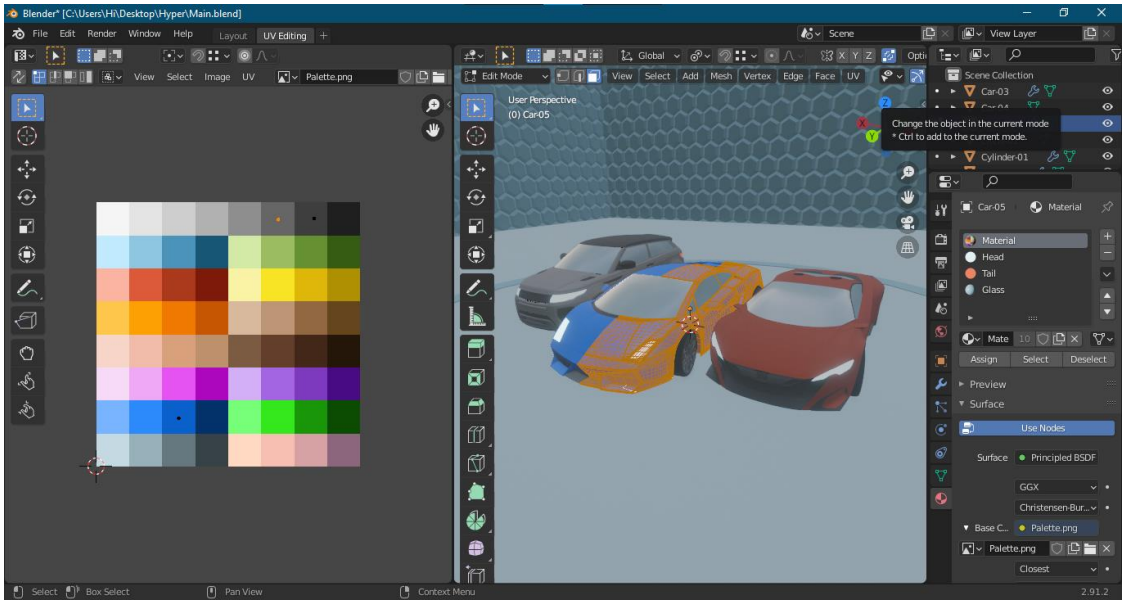

Blender* [C:\Users\H\Desktop\Hyperdrive.blend]
File Edit Render Window Help
Layout
Scene
View Layer
Object Mode View Select Add Object
User Perspective
(0) Scene Collection | Track-02
Objects 1 / 3
Vertices 3,767
Edges 7,313
Faces 3,554
Triangles 7,026
Move
Move X 0 m
Y 0 m
Z 0.77582 m
Orientation Global
Proportional Editing
Track-02
Material
Preview
Surface
Volume
Settings
Playback Keying View Marker
Start 1 End 250
Select Box Select Rotate View Object Context Menu
2.91.2


Blender* [C:\Users\H\Desktop\Hyperdrive.blend]
File Edit Render Window Help
Layout
Scene
View Layer
Object Mode View Select Add Object
User Perspective
(0) Scene Collection | Car-01
Objects 1 / 1
Vertices 2,169
Edges 4,206
Faces 2,044
Triangles 4,104
Scene Collection
Car-01
Material
Head
Tail
Glass
Preview
Surface
Playback Keying View Marker
Start 1 End 250
Select Box Select Rotate View Object Context Menu
2.91.2


Blender* [C:\Users\H\Desktop\Hyperdrive.blend]
File Edit Render Window Help
Layout
UV Editing
Scene
View Layer
View Select Image UV
Palette.png
Edit Mode View Select Add Mesh Vertex Edge Face UV
User Perspective
(0) Car-02
Scene Collection
Car-02
Mesh
Vertex Groups
Shape Keys
UV Maps
Vertex Colors
Face Maps
Normals
Texture Space
Remesh
Geometry Data
Custom Properties
Change Frame Box Select Pan View UV Context Menu
2.91.2


Blender* [C:\Users\H\Desktop\Hyperdrive.blend]
File Edit Render Window Help
Layout
Scene
View Layer
Edit Mode
View Select Add Mesh Vertex Edge Face UV
User Perspective
(0) Car-05
Objects 1 / 1
Vertices 1,969 / 26,361
Edges 3,812 / 50,773
Faces 1,845 / 24,559
Triangles 48,923
Select Linked
Deselect
Delimit
Normal
Material
Seam
Sharp
UVs
Scene Collection
Car-05
Material
Head
Tail
Glass
Assign
Select
Deselect
Preview
Playback Keying View Marker
Start 1 End 250
Select Box Select Rotate View Call Menu
2.91.2


Blender* [C:\Users\H\Desktop\Hyperdrive.blend]
File Edit Render Window Help
Layout
Scene
View Layer
Object Mode
View Select Add Object
User Perspective
(0) Scene Collection | Wheel-01
Objects 1 / 7
Vertices 1,087
Edges 2,116
Faces 1,036
Triangles 2,076
Wheel-01
Material
Preview
Surface
Volume
Settings
Playback Keying View Marker
Start 1 End 250
Select Box Select Rotate View Object Context Menu
2.91.2


Blender* [C:\Users\H\Desktop\Hyper\Main.blend]
File Edit Render Window Help
Layout
UV Editing
Scene
View Layer
View Select Image UV
Palette.png
Edit Mode
View Select Add Mesh Vertex Edge Face UV
User Perspective
(0) Car-05
Change the object in the current mode
* Ctrl to add to the current mode.
Scene Collection
Car-03
Cylinder-01
Car-05
Material
Head
Tail
Glass
Assign
Select
Deselect
Preview
Surface
Use Nodes
Surface
Principled BSDF
GGX
Christensen-Bur...
Base C...
Palette.png
Closest
Select Box Select Pan View Context Menu
2.91.2

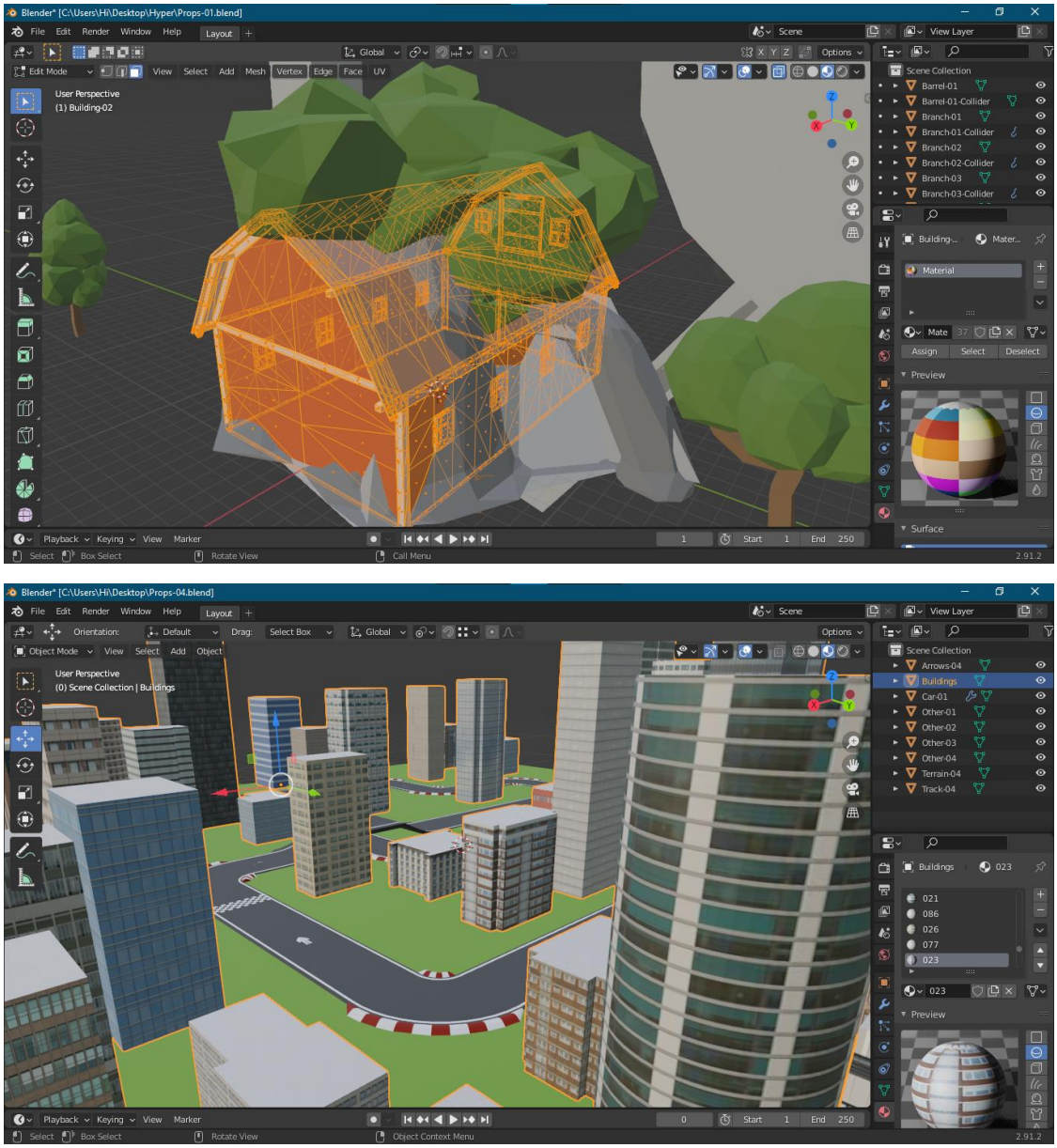


Figure 22 - Hyperdrive Models in Blender

4.4 - Database Design

Hyperdrive uses Unity PlayerPrefs to store data such as unlocked levels and cars. PlayerPrefs is a class that stores Player preferences between game sessions. It can store string, float, and integer values into the user's platform registry. Windows Registry is a hierarchical database that contains information, settings, options, and other values for programs and hardware installed on all versions of Microsoft Windows operating systems.

Unity stores PlayerPrefs data differently based on which operating system the application runs on. In the file paths given on this page, the company name and product name are the names set in Unity's Player Settings. On Windows, PlayerPrefs are stored in `HKCU\Software\CompanyName\ProductName` key.

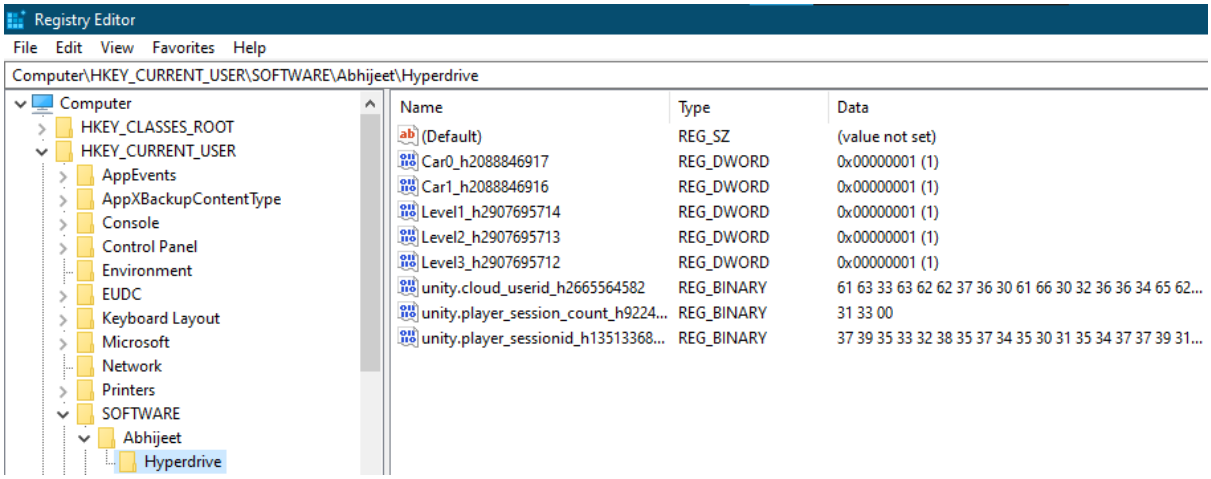
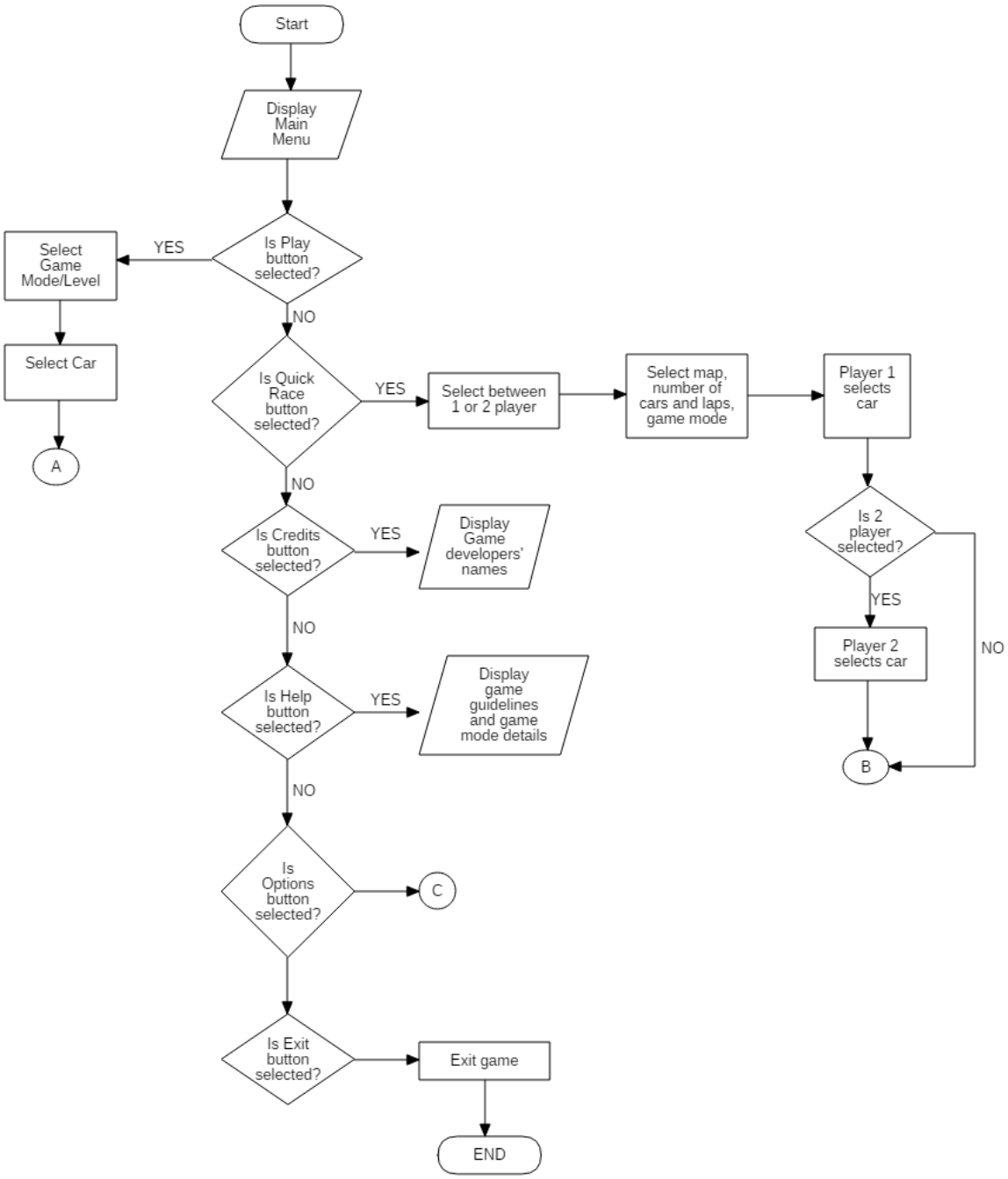
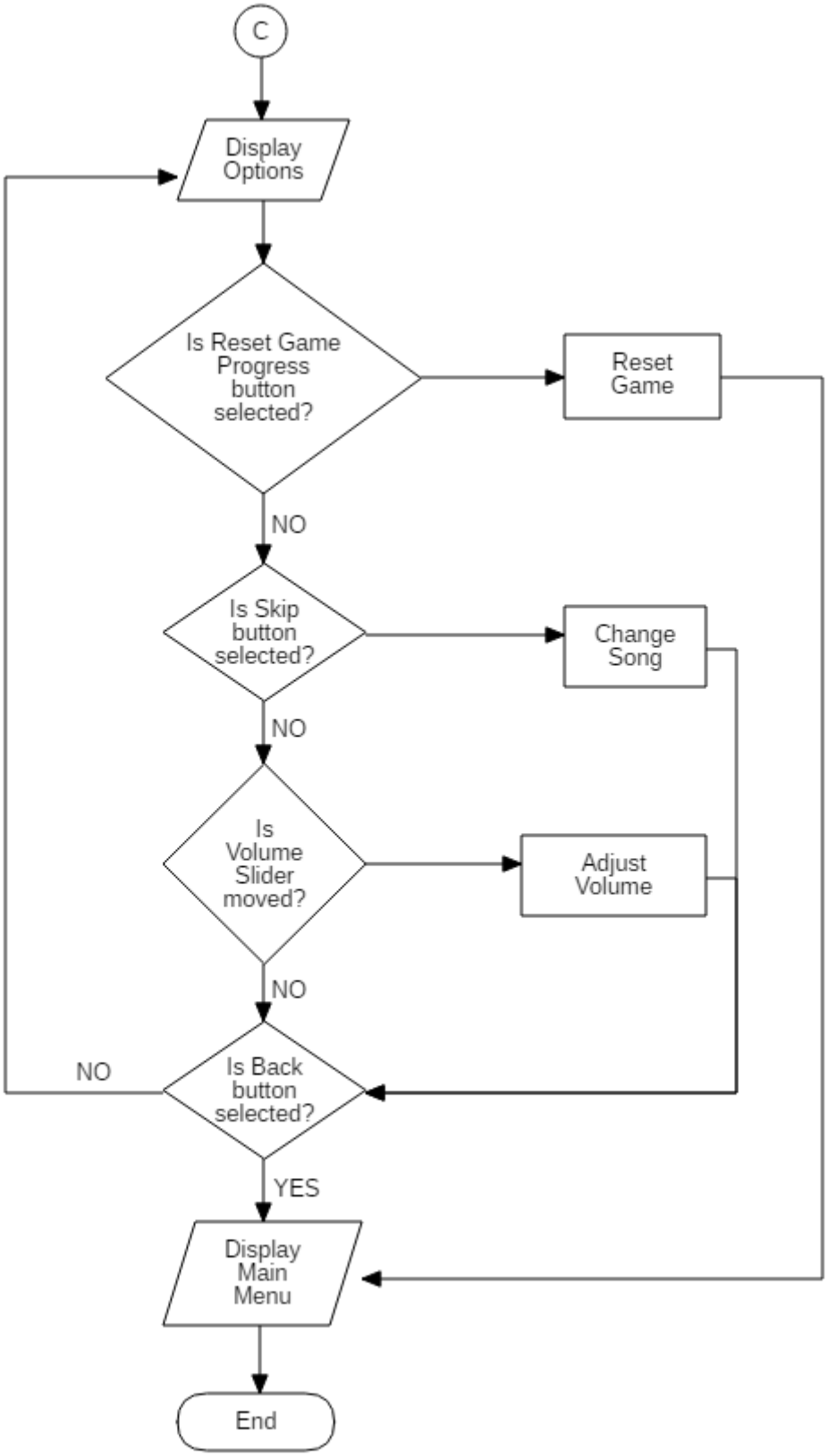


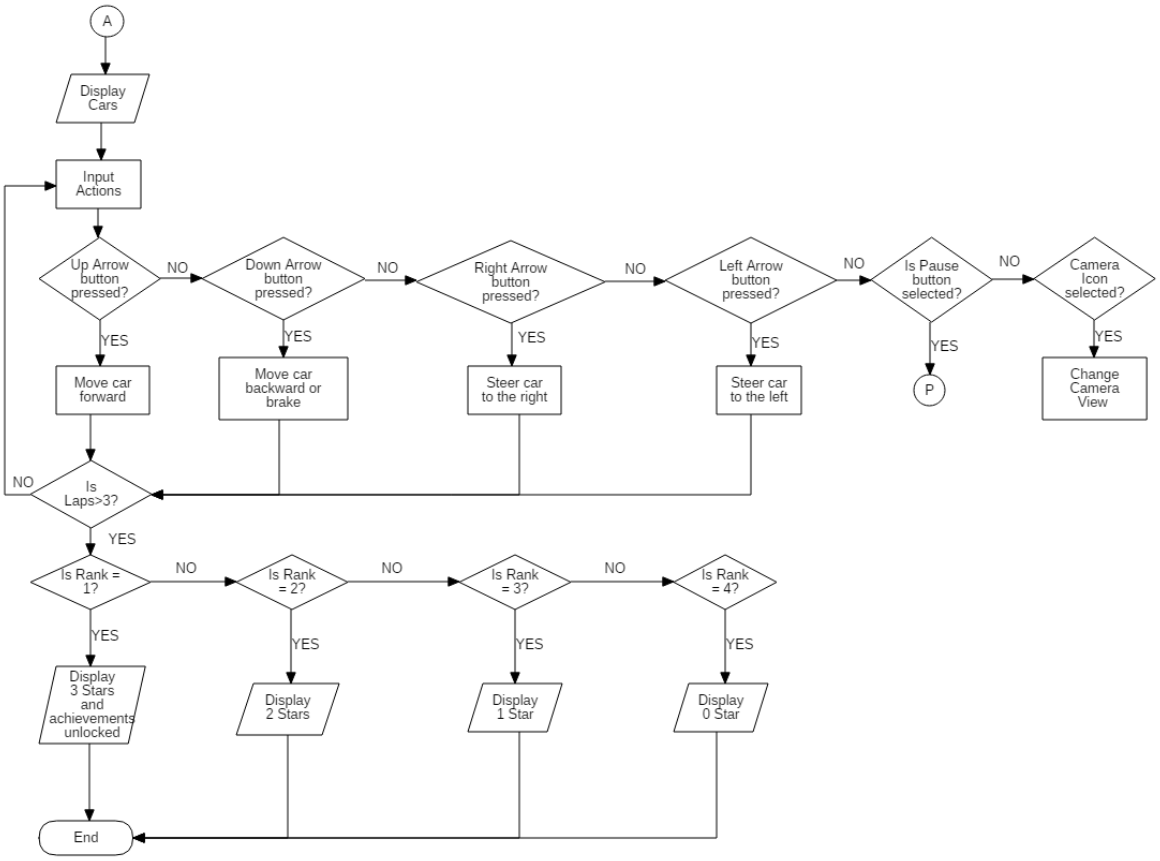
Figure 23 - Windows Registry

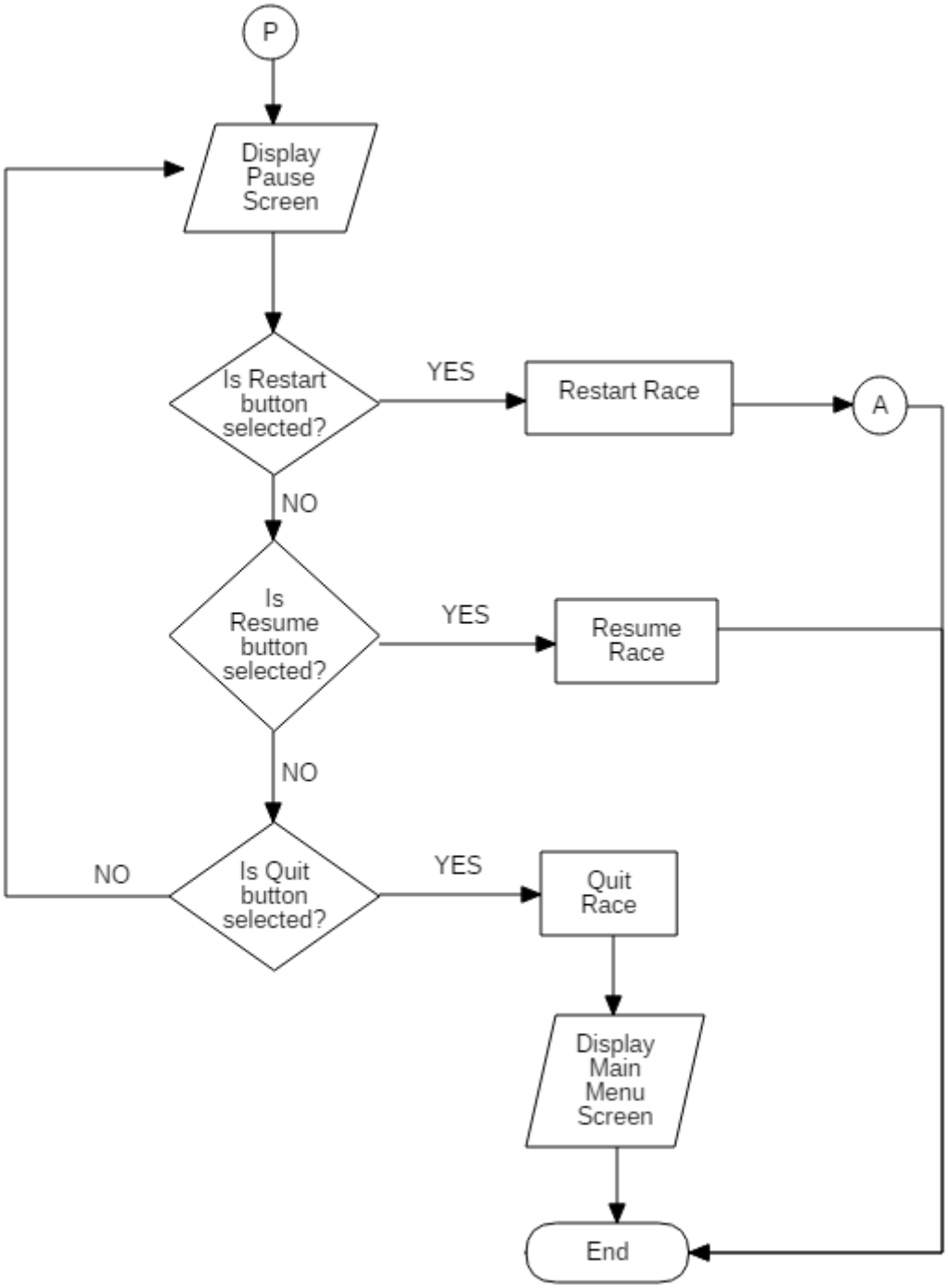
4.5 - Program Design

4.5.1 - Flowchart




C
Display Options
Is Reset Game Progress button selected?
Reset Game
NO
Is Skip button selected?
Change Song
NO
Is Volume Slider moved?
Adjust Volume
NO
Is Back button selected?
NO
YES
Display Main Menu
End


A
Display Cars
Input Actions
Up Arrow button pressed?
NO
Down Arrow button pressed?
NO
Right Arrow button pressed?
NO
Left Arrow button pressed?
NO
Is Pause button selected?
NO
Camera Icon selected?
YES
Move car forward
YES
Move car backward or brake
YES
Steer car to the right
YES
Steer car to the left
YES
P
YES
Change Camera View
NO
Is Laps>3?
YES
Is Rank = 1?
NO
Is Rank = 2?
NO
Is Rank = 3?
NO
Is Rank = 4?
YES
Display 3 Stars and achievements unlocked
YES
Display 2 Stars
YES
Display 1 Star
YES
Display 0 Star
End


P
Display Pause Screen
Is Restart button selected?
YES
Restart Race
A
NO
Is Resume button selected?
YES
Resume Race
NO
Is Quit button selected?
NO
YES
Quit Race
Display Main Menu Screen
End

5 - Implementation and Testing

5.1 - System Requirements

Operating system version	Windows 7 (SP1+) and Windows 10
RAM	1024 MB+
Hard disk space	512 MB+
CPU	x86, x64 architecture with SSE2 instruction set support
Graphics API	DX10, DX11, DX12 capable
Additional requirements	Mouse, keyboard, sound card, hardware vendor officially supported drivers

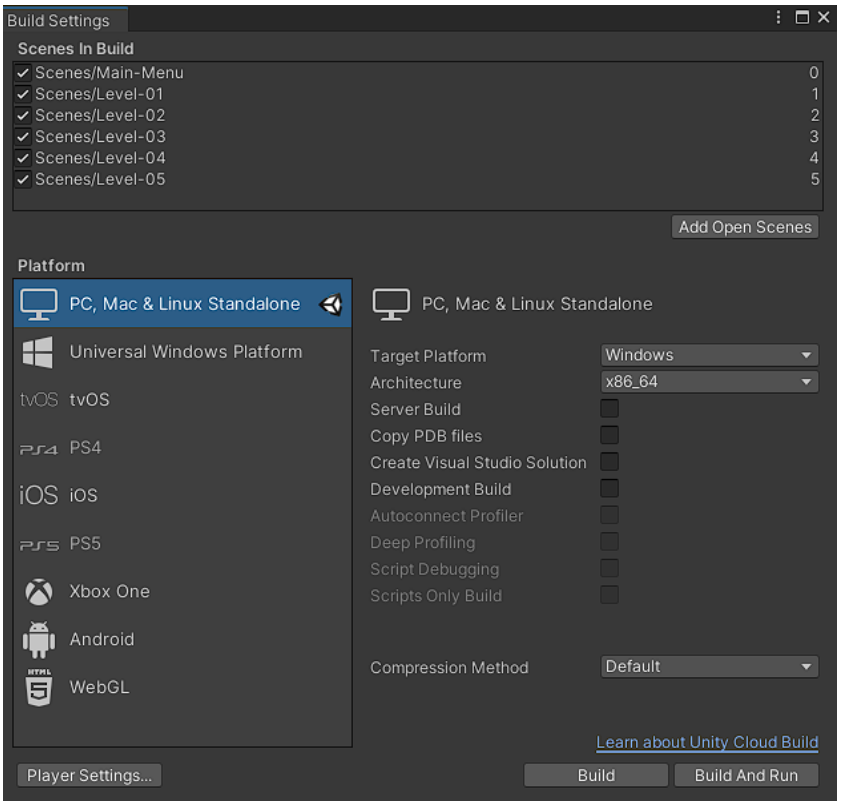


Figure 24 - Unity Build Settings

5.2 - Implementation of Each Component

The entire source code of the game is too long to be fully displayed here. The following are *extracts* of the *most important* components. **Several lines of code have been omitted. Assume all required namespaces (such as `using UnityEngine;`) and variables (such as `int num = 0;`) have been appropriately declared and initialised.** Also keep in mind that reading code is *significantly harder* than writing it.

5.2.1 - Script for Controlling each Player in the Race (Player.cs)

The component is attached to each car in the race. Each instance of this class first fetches attributes such as the race mode, control type, difficulty type and number of laps, from the static GameManager class. Every frame, it checks for human player input if the control type is human, or controls the car if the control type is set to robot. It updates the waypoints (path followed by the AI) and the car's current rank in real-time. Depending on the race mode, it also holds the timer and eliminates or infects the car as required. Every 3 seconds, this script checks if the car is falling off the map or if the robot is stuck and then resets its position to the last checkpoint passed.

```
public class Player : MonoBehaviour
{
    private void Start()
    {
        gameType = GameManager.instance.gameType;
        raceMode = GameManager.instance.raceMode;
        difficultyType = GameManager.instance.difficultyType;
        switch(difficultyType)
        {
            case Menu.DifficultyTypes.Easy: maxRobotThrottle = 0.7f; robotStee
r = 1.5f; break;
            case Menu.DifficultyTypes.Hard: maxRobotThrottle = 1.5f; robotStee
r = 2.5f; break;
            default: maxRobotThrottle = 1.1f; robotSteer = 2f; break;
        }
        robotThrottle = maxRobotThrottle - 0.2f;
        minRobotThrottle = 0.5f;
        numLaps = GameManager.instance.numLaps;
        numCars = GameManager.instance.numCars;
        raceTimeStamp = Time.time;
        foreach (Transform car in carsParent.transform)
        {
            Player player = car.GetComponent<Player>();
            if ((player != null) && (car.gameObject.activeSelf))
            { cars.Add(player); }
        }
        StartLap();
        StartCoroutine(IntervalUpdate());
        StartCoroutine(UpdateRank());
```

```
        if ((raceMode == Menu.RaceModes.Escape) || (raceMode == Menu.RaceModes
.Survival) || (raceMode == Menu.RaceModes.Infection) || (raceMode == Menu.Race
Modes.Special))
        { StartCoroutine(ModeTimer()); }
        carName = gameObject.name;
        if ((gameType != Menu.GameTypes.quickRace2P) && (controlType == Player
.ControlTypes.human))
        { carName += " (You)"; }
        else if ((gameType == Menu.GameTypes.quickRace2P) && (controlType == P
layer.ControlTypes.human))
        { carName += " (A)"; }
        else if ((gameType == Menu.GameTypes.quickRace2P) && (controlType == P
layer.ControlTypes.humanB))
        { carName += " (B)"; }
    }
    private void StartLap()
    {
        currentLap++;
        previousCheckpoint = 1;
        lapTimeStamp = Time.time;
    }
    private void EndLap()
    {
        lastLapTime = Time.time - lapTimeStamp;
        if (lastLapTime > 599.99f) { lastLapTime = 599.99f; }
        bestLapTime = Mathf.Min(lastLapTime, bestLapTime);
        if (bestLapTime > 599.99f) { bestLapTime = 599.99f; }
        currentWaypoint = waypoints[0];
        waypointProgess = 0;
    }
    private void OnTriggerEnter(Collider collider)
    {
        if (collider.gameObject.layer != checkpointLayer) { return; }
        colliderName = collider.gameObject.name;
        if (colliderName == "1")
        {
            if (previousCheckpoint == checkpointCount)
            { EndLap(); }
            if (currentLap == 0 || previousCheckpoint == checkpointCount)
            { StartLap(); }
            if (currentLap == numLaps + 1)
            { raceOver = true; }
            if ((raceOver) && (controlType == ControlTypes.human) && (gameType
 != Menu.GameTypes.quickRace2P))
            {
                GameManager.instance.UIController.ShowRaceOver();
                GameManager.instance.UIController.music.audioSource.PlayOneSho
t(GameManager.instance.UIController.finishSound);
```

```
                GameManager.instance.UIController.confetti.Play();
            }
            else if ((raceOver) && (controlType == ControlTypes.human) && (gam
eType == Menu.GameTypes.quickRace2P))
            {
                StartCoroutine(GameManager.instance.UIController.ShowRaceOverA
());
                GameManager.instance.UIController.music.audioSource.PlayOneSho
t(GameManager.instance.UIController.finishSound);
                GameManager.instance.UIController.confetti.Play();
            }
            else if ((raceOver) && (controlType == ControlTypes.humanB) && (ga
meType == Menu.GameTypes.quickRace2P))
            {
                StartCoroutine(GameManager.instance.UIController.ShowRaceOverB
());
                GameManager.instance.UIController.music.audioSource.PlayOneSho
t(GameManager.instance.UIController.finishSound);
                GameManager.instance.UIController.confettiB.Play();
            }
            return;
        }
        if (colliderName == (previousCheckpoint + 1).ToString())
        { previousCheckpoint++; }
    }
    private void Update()
    {
        UpdateWaypoint();
        if (controlType == ControlTypes.human) { HumanControl(); }
        else if (controlType == ControlTypes.humanB) { HumanControlB(); }
        else if (controlType == ControlTypes.robot) { RobotControl(); }
        if (currentLapTime < 599.99f)
        { currentLapTime = Time.time - lapTimeStamp; }
        else
        { currentLapTime = 599.99f; }
        if ((!raceOver) && (totalRaceTime < 599.99f))
        { totalRaceTime = Time.time - raceTimeStamp; }
        if (Input.GetKeyDown(KeyCode.Return) && (controlType == Player.Control
Types.human) && (!GameManager.instance.UIController.racePaused) && (totalRaceT
ime > 3f) && (!recentReset))
        { StartCoroutine(ResetCarPosition()); recentReset = true; StartCorouti
ne(UpdateRecentReset()); }
        else if (Input.GetKeyDown(KeyCode.Tab) && (controlType == Player.Contr
olTypes.humanB) && (!GameManager.instance.UIController.racePaused) && (totalRa
ceTime > 3f) && (!recentReset))
        { StartCoroutine(ResetCarPosition()); recentReset = true; StartCorouti
ne(UpdateRecentReset()); }
    }
```

```
    private void HumanControl()
    {
        carController.steer = GameManager.instance.inputController.steerInput;
        carController.throttle = GameManager.instance.inputController.throttle
Input;
    }
    private void HumanControlB()
    {
        carController.steer = GameManager.instance.inputControllerB.steerInput
B;
        carController.throttle = GameManager.instance.inputControllerB.throttl
eInputB;
    }
    private void RobotControl()
    {
        waypointRelativeDistance = transform.InverseTransformPoint(currentWayp
oint.position);
        waypointRelativeDistance /= waypointRelativeDistance.magnitude;
        carController.steer = (waypointRelativeDistance.x / waypointRelativeDi
stance.magnitude) * robotSteer;
        carController.throttle = robotThrottle;
    }
    private void UpdateWaypoint()
    {
        waypointPosition = transform.position;
        waypointDistance = Mathf.Infinity;
        for (int i = 0; i < waypoints.Count; i++)
        {
            waypointDistanceDifference = waypoints[i].position - waypointPosit
ion;
            currentWaypointDistance = waypointDistanceDifference.magnitude;
            if ((currentWaypointDistance < waypointDistance) && (i >= waypoint
Progess) && (i <= waypointProgess + 5) && (currentWaypoint != waypoints[waypoi
nts.Count - 1]))
            {
                currentWaypoint = waypoints[i + waypointDistanceOffset];
                waypointDistance = currentWaypointDistance;
                waypointProgess = i;
            }
        }
    }
    private IEnumerator UpdateRank()
    {
        WaitForSeconds delay = new WaitForSeconds(0.1f);
        int tempCarRank;
        int otherCarCurrentLap;
        int thisCarCurrentWaypoint;
        int otherCarCurrentWaypoint;
```

```
        float thisCarCurrentWaypointDistance;
        float otherCarCurrentWaypointDistance;
        carRank = 1;
        while (!raceOver)
        {
            yield return delay;
            tempCarRank = 1;
            for (int i = 0; i < cars.Count; i++)
            {
                if ((gameObject.GetInstanceID() == cars[i].gameObject.GetInsta
nceID()) || (cars[i].carEliminated))
                { continue; }
                otherCarCurrentLap = cars[i].currentLap;
                if (currentLap < otherCarCurrentLap) { tempCarRank++; }
                else if (currentLap == otherCarCurrentLap)
                {
                    thisCarCurrentWaypoint = int.Parse(currentWaypoint.gameObj
ect.name);
                    otherCarCurrentWaypoint = int.Parse(cars[i].currentWaypoin
t.gameObject.name);
                    if (thisCarCurrentWaypoint < otherCarCurrentWaypoint) { te
mpCarRank++; }
                    else if (thisCarCurrentWaypoint == otherCarCurrentWaypoint
)
                    {
                        thisCarCurrentWaypointDistance = (currentWaypoint.posi
tion - transform.position).sqrMagnitude;
                        otherCarCurrentWaypointDistance = (cars[i].currentWayp
oint.position - cars[i].transform.position).sqrMagnitude;
                        if (thisCarCurrentWaypointDistance > otherCarCurrentWa
ypointDistance) { tempCarRank++; }
                    }
                }
            }
            carRank = tempCarRank;
        }
    }
    private IEnumerator ModeTimer()
    {
        int timerTime;
        int timerTimeLeft;
        WaitForSeconds oneSecond = new WaitForSeconds(1f);
        timerTime = 35;
        yield return new WaitForSeconds(0.1f);
        while (!raceOver)
        {
            timerTimeLeft = timerTime;
            lapProgess = currentLap;
```

```
            while (timerTimeLeft > 0)
            {
                if (timerTimeLeft >= 10)
                { GameManager.instance.UIController.timerText.text = "0:" + ti
merTimeLeft.ToString(); }
                else
                { GameManager.instance.UIController.timerText.text = "0:0" + t
imerTimeLeft.ToString(); }
                if ((timerTimeLeft == 3) && (controlType == ControlTypes.human
))
                { GameManager.instance.UIController.music.audioSource.PlayOneS
hot(GameManager.instance.UIController.timerSound); }
                yield return oneSecond;
                timerTimeLeft--;
            }
            if (raceOver) { break; }
            ModeTimerOver();
            if (timerTime > 25) { timerTime -= 5; }
        }
    }
    private void ModeTimerOver()
    {
        switch (raceMode)
        {
            case Menu.RaceModes.Escape:
            {
                if (lapProgess == currentLap)
                { EliminateCar(); }
                break;
            }
            case Menu.RaceModes.Survival:
            {
                if (carRank == numCars)
                { EliminateCar(); }
                break;
            }
            case Menu.RaceModes.Infection:
            {
                if (carRank == numCars)
                { StartCoroutine(InfectCar()); }
                break;
            }
            case Menu.RaceModes.Special:
            {
                if ((lapProgess == currentLap) || (carRank == numCars))
                { EliminateCar(); }
                break;
            }
```

```
        }
        numCars = GameManager.instance.UIController.numCars;
    }
    private void EliminateCar()
    {
        string infoString;
        raceOver = true;
        carEliminated = true;
        GameManager.instance.UIController.numCars--;
        if (controlType == Player.ControlTypes.robot)
        { StartCoroutine(EliminateRobot()); }
        else if ((controlType == Player.ControlTypes.human) && (gameType != Me
nu.GameTypes.quickRace2P))
        {
            GameManager.instance.UIController.music.audioSource.PlayOneShot(Ga
meManager.instance.UIController.eliminateSound);
            GameManager.instance.UIController.ShowRaceOver();
        }
        else if ((controlType == Player.ControlTypes.human) && (gameType == Me
nu.GameTypes.quickRace2P))
        {
            GameManager.instance.UIController.music.audioSource.PlayOneShot(Ga
meManager.instance.UIController.eliminateSound);
            StartCoroutine(GameManager.instance.UIController.ShowRaceOverA());
        }
        else if ((controlType == Player.ControlTypes.humanB) && (gameType == M
enu.GameTypes.quickRace2P))
        {
            GameManager.instance.UIController.music.audioSource.PlayOneShot(Ga
meManager.instance.UIController.eliminateSound);
            StartCoroutine(GameManager.instance.UIController.ShowRaceOverB());
        }
        infoString = carName + " ELIMINATED";
        if (GameManager.instance.UIController.showingInfo)
        { StartCoroutine(GameManager.instance.UIController.DelayedShowInfo(inf
oString)); }
        else
        { GameManager.instance.UIController.ShowInfo(infoString); }
    }
    private IEnumerator EliminateRobot()
    {
        controlType = Player.ControlTypes.idle;
        carController.steer = 0f;
        carController.throttle = 0f;
        yield return new WaitForSeconds(10f);
        gameObject.SetActive(false);
    }
    private IEnumerator InfectCar()
```

```
    {
        string infoString;
        carInfected = true;
        carController.maxThrottle += 1500f;
        infectionSmoke.Play();
        if (controlType == Player.ControlTypes.human)
        {
            GameManager.instance.UIController.music.audioSource.PlayOneShot(Ga
meManager.instance.UIController.infectSound);
            GameManager.instance.UIController.IncreaseFieldOfView();
        }
        else if (controlType == Player.ControlTypes.humanB)
        {
            GameManager.instance.UIController.music.audioSource.PlayOneShot(Ga
meManager.instance.UIController.infectSound);
            GameManager.instance.UIController.IncreaseFieldOfViewB();
        }
        infoString = carName + " INFECTED";
        if (GameManager.instance.UIController.showingInfo)
        { StartCoroutine(GameManager.instance.UIController.DelayedShowInfo(inf
oString)); }
        else
        { GameManager.instance.UIController.ShowInfo(infoString); }
        yield return new WaitForSeconds(10f);
        DisinfectCar();
    }
    private void DisinfectCar()
    {
        string infoString;
        carController.maxThrottle -= 1500f;
        infectionSmoke.Stop();
        if (gameObject.GetInstanceID() == GameManager.instance.UIController.hu
manCar.gameObject.GetInstanceID())
        {
            GameManager.instance.UIController.music.audioSource.PlayOneShot(Ga
meManager.instance.UIController.disinfectSound);
            GameManager.instance.UIController.DecreaseFieldOfView();
        }
        else if ((gameType == Menu.GameTypes.quickRace2P) && (gameObject.GetIn
stanceID() == GameManager.instance.UIController.humanCarB.gameObject.GetInstan
ceID()))
        {
            GameManager.instance.UIController.music.audioSource.PlayOneShot(Ga
meManager.instance.UIController.disinfectSound);
            GameManager.instance.UIController.DecreaseFieldOfViewB();
        }
        infoString = carName + " DISINFECTED";
        if (GameManager.instance.UIController.showingInfo)
```

```
        { StartCoroutine(GameManager.instance.UIController.DelayedShowInfo(inf
oString)); }
        else
        { GameManager.instance.UIController.ShowInfo(infoString); }
        carInfected = false;
    }
    private void OnCollisionEnter(Collision collider)
    {
        if ((collider.gameObject.layer == carLayer) || (collider.gameObject.la
yer == propsLayer))
        { audioSource.PlayOneShot(impactSound, 1f); }
        if ((raceMode != Menu.RaceModes.Infection) || (collider.gameObject.lay
er != carLayer) || (raceOver) || (carInfected))
        { return; }
        Player otherCar = collider.gameObject.GetComponent<Player>();
        if (otherCar.carInfected) { StartCoroutine(InfectCar()); }
    }
    private IEnumerator IntervalUpdate()
    {
        while (true)
        {
            yield return delay;
            if (transform.position.y <= -
20f) { StartCoroutine(ResetCarPosition()); }
            if ((controlType == ControlTypes.robot) && ((carBody.velocity.sqrM
agnitude < 4f) || (currentWaypoint == waypoints[waypoints.Count - 1])))
            { StartCoroutine(CheckStuckRobot()); }
            robotThrottle = Random.Range(minRobotThrottle, maxRobotThrottle);
        }
    }
    private IEnumerator CheckStuckRobot()
    {
        yield return delay;
        if ((carBody.velocity.sqrMagnitude < 4f) || (currentWaypoint == waypoi
nts[waypoints.Count - 1]))
        { StartCoroutine(ResetCarPosition()); }
    }
    private IEnumerator ResetCarPosition()
    {
        GetResetPosition();
        if (controlType == ControlTypes.robot)
        { yield return new WaitForSeconds(Random.Range(0, 3)); }
        transform.position = resetCheckpoint.position;
        transform.rotation = resetCheckpoint.rotation;
        carBody.velocity = Vector3.zero;
        carBody.angularVelocity = Vector3.zero;
    }
    private void GetResetPosition()
```

```
    {
        if ((controlType == ControlTypes.human) || (controlType == ControlType
s.humanB))
        { resetCheckpoint = checkpointsParent.Find(previousCheckpoint.ToString
()); return; }
        if (previousCheckpoint != checkpointCount)
        { resetCheckpoint = checkpointsParent.Find((previousCheckpoint + 1).To
String()); return; }
        resetCheckpoint = checkpointsParent.Find("1");
    }
}
```

5.2.2 - Script for Controlling the UI during the Race (UIController.cs)

This component is attached as a child of the GameManager. It activates and deactivates canvases (countdown, race, pause, results canvases, etc) as appropriate. It updates the race canvas (rank, speed, lap time, etc) every frame. At the end of the race, it stores data such as any unlocked levels or cars in Windows Registry using Unity PlayerPrefs.

```
public class UIController : MonoBehaviour
{
    private IEnumerator Start()
    {
        animating = true;
        loadingCanvas.gameObject.SetActive(true);
        buttonsCanvas.gameObject.SetActive(false);
        buttonsCanvas2P.gameObject.SetActive(false);
        countdownCanvas.gameObject.SetActive(false);
        overCanvas.gameObject.SetActive(false);
        pauseCanvas.gameObject.SetActive(false);
        timerCanvas.gameObject.SetActive(false);
        humanCar = GameManager.instance.humanCar.GetComponent<Player>();
        gameType = GameManager.instance.gameType;
        difficultyType = GameManager.instance.difficultyType;
        mapName = GameManager.instance.mapName;
        raceMode = GameManager.instance.raceMode;
        numLaps = GameManager.instance.numLaps;
        numCars = GameManager.instance.numCars;
        selectedLevel = GameManager.instance.selectedLevel;
        if (selectedLevel == 5) { Physics.gravity = new Vector3(0f, -
15f, 0f); }
        else { Physics.gravity = new Vector3(0f, -30f, 0f); }
        if (gameType == Menu.GameTypes.quickRace2P)
        {
            raceOverB = false;
            raceCanvasA.alpha = 0f;
            raceCanvasA.gameObject.SetActive(true);
            raceCanvasB.alpha = 0f;
```

```
            raceCanvasB.gameObject.SetActive(true);
            columnCanvas.gameObject.SetActive(true);
            humanCarB = GameManager.instance.humanCarB.GetComponent<Player>();
        }
        else
        {
            raceOverB = true;
            raceCanvas.alpha = 0f;
            raceCanvas.gameObject.SetActive(true);
            columnCanvas.gameObject.SetActive(false);
        }
        cars = new List<Player>(humanCar.cars);
        infoDisappearTime = Mathf.Infinity;
        delayedInfoAppear = 0f;
        if ((raceMode == Menu.RaceModes.Escape) || (raceMode == Menu.RaceModes
.Survival) || (raceMode == Menu.RaceModes.Infection) || (raceMode == Menu.Race
Modes.Special))
        {
            infoCanvas.gameObject.SetActive(true);
            timerCanvas.alpha = 0f;
            timerCanvas.gameObject.SetActive(true);
        }
        else
        {
            infoCanvas.gameObject.SetActive(false);
            timerCanvas.gameObject.SetActive(false);
        }
        switch (gameType)
        {
            case Menu.GameTypes.quickRace1P: pauseText[0].text = "1 Player"; b
reak;
            case Menu.GameTypes.quickRace2P: pauseText[0].text = "2 Players";
break;
            default: pauseText[0].text = ("Level " + selectedLevel.ToString())
; break;
        }
        pauseText[1].text = mapName.ToString();
        pauseText[2].text = raceMode.ToString();
        pauseText[3].text = difficultyType.ToString();
        yield return new WaitForSecondsRealtime(0.1f);
        StartCoroutine(FadeOut(loadingCanvas));
        StartCoroutine(StartCountdownTimer());
        StartCoroutine(UpdateSpeed());
        if (numCars > 1)
        { music.audioSource.PlayOneShot(multipleRevSound); }
        else
        { music.audioSource.PlayOneShot(singleRevSound); }
        windowOpen = false;
```

```
    }
    private void LateUpdate()
    {
        if ((humanCar == null) || ((humanCarB == null) && (gameType == Menu.Ga
meTypes.quickRace2P)))
        { return; }
        if (gameType == Menu.GameTypes.quickRace2P)
        { UpdateRacePanelA(); UpdateRacePanelB(); }
        else
        { UpdateRacePanel(); }
        if ((raceOver) && (gameType != Menu.GameTypes.quickRace2P))
        { UpdateLeaderboardTime(ref leaderboardTime); }
        else if ((raceOver) && (raceOverB) && (gameType == Menu.GameTypes.quic
kRace2P) && (leaderboardCanvasA.gameObject.activeSelf))
        { UpdateLeaderboardTime(ref leaderboardTimeA); }
        else if ((raceOver) && (raceOverB) && (gameType == Menu.GameTypes.quic
kRace2P) && (leaderboardCanvasB.gameObject.activeSelf))
        { UpdateLeaderboardTime(ref leaderboardTimeB); }
        if (Time.unscaledTime >= infoDisappearTime)
        {
            infoCanvas.LeanAlpha(0f, halfSecond).setEaseInOutQuart().setIgnore
TimeScale(true);
            infoDisappearTime = Mathf.Infinity;
            showingInfo = false;
        }
        if (animating || windowOpen) { return; }
        if (Input.GetKeyDown(KeyCode.Escape))
        {
            if ((raceStarted) && (!cameraSwirl) && (racePaused)) { ResumeRace(
); }
            else if ((raceStarted) && (!cameraSwirl) && (!racePaused)) { Pause
Race(); }
            else if ((gameType == Menu.GameTypes.quickRace2P) && (raceStarted)
 && (racePaused) && ((!cameraSwirl) || (!cameraSwirlB)))
            { ResumeRace(); }
            else if ((gameType == Menu.GameTypes.quickRace2P) && (raceStarted)
 && (!racePaused) && ((!cameraSwirl) || (!cameraSwirlB)))
            { PauseRace(); }
            if ((raceOver) && (raceOverB)) { BackToMenuWrapper(); }
        }
        if (Input.GetKeyDown(KeyCode.Return) && (raceOver) && (raceOverB)) { B
ackToMenuWrapper(); }
    }
    private void UpdateRacePanel()
    {
        if (humanCar.currentLap != currentLap || currentLap == 0)
        {
            currentLap = humanCar.currentLap;
```

```
            if (!humanCar.raceOver) { currentLapText[0].text = $"{currentLap}
of {numLaps}"; }
        }
        if ((humanCar.carRank != carRank) || (humanCar.numCars != numCars))
        {
            carRank = humanCar.carRank;
            humanCar.numCars = numCars;
            carRankText[0].text = $"{carRank} of {numCars}";
        }
        if (humanCar.currentLapTime != currentLapTime)
        {
            currentLapTime = humanCar.currentLapTime;
            currentTimeText[0].text = $"{(int)currentLapTime/60}:{(currentLapT
ime)%60:00.00}";
        }
        if (humanCar.lastLapTime != lastLapTime)
        {
            lastLapTime = humanCar.lastLapTime;
            lastLapTimeText[0].text = $"{(int)lastLapTime/60}:{(lastLapTime)%6
0:00.00}";
        }
        if (humanCar.bestLapTime != bestLapTime)
        {
            bestLapTime = humanCar.bestLapTime;
            bestLapTimeText[0].text = bestLapTime <= 599.99f ? $"{(int)bestLap
Time/60}:{(bestLapTime)%60:00.00}" : "0:00.00";
        }
    }
    private IEnumerator StartCountdownTimer()
    {
        animating = true;
        WaitForSecondsRealtime delay = new WaitForSecondsRealtime(1f);
        FadeIn(countdownCanvas);
        Time.timeScale = 0f;
        raceStarted = false;
        countdownTime = 3;
        while (countdownTime > 0)
        {
            countdownText.text = countdownTime.ToString();
            music.audioSource.PlayOneShot(countdownSound);
            yield return delay;
            countdownTime--;
            if (!raceOver) { cameraSwirl = false; }
            if (!raceOverB) { cameraSwirlB = false; }
        }
        countdownText.text = "GO";
        music.audioSource.PlayOneShot(goSound);
        Time.timeScale = 1f;
```

```
        if (gameType == Menu.GameTypes.quickRace2P)
        {
            raceCanvasA.LeanAlpha(1f,halfSecond).setEaseInOutQuart().setIgnore
TimeScale(true);
            raceCanvasB.LeanAlpha(1f,halfSecond).setEaseInOutQuart().setIgnore
TimeScale(true);
            FadeIn(buttonsCanvas2P);
        }
        else
        {
            raceCanvas.LeanAlpha(1f,halfSecond).setEaseInOutQuart().setIgnoreT
imeScale(true);
            FadeIn(buttonsCanvas);
        }
        if ((raceMode == Menu.RaceModes.Escape) || (raceMode == Menu.RaceModes
.Survival) || (raceMode == Menu.RaceModes.Infection) || (raceMode == Menu.Race
Modes.Special))
        { timerCanvas.LeanAlpha(1f,halfSecond).setEaseInOutQuart().setIgnoreTi
meScale(true); }
        yield return delay;
        StartCoroutine(FadeOut(countdownCanvas));
        if ((raceOver) && (gameType == Menu.GameTypes.quickRace2P))
        { FadeIn(overCanvasA); FadeIn(resultsCanvasA); }
        else if ((raceOverB) && (gameType == Menu.GameTypes.quickRace2P))
        { FadeIn(overCanvasB); FadeIn(resultsCanvasB); }
        racePaused = false;
        raceStarted = true;
    }
    private IEnumerator UpdateSpeed()
    {
        WaitForSeconds delay = new WaitForSeconds(0.1f);
        while ((!raceOver) || (!raceOverB))
        {
            carSpeed =  new Vector3(humanCar.carBody.velocity.x, 0f, humanCar.
carBody.velocity.z).magnitude * 3.6f;
            if (carSpeed > 999f) { carSpeed = 999f; }
            if (gameType == Menu.GameTypes.quickRace2P)
            {
                carSpeedText[1].text = $"{(int)carSpeed} km/h";
                carSpeedB =  new Vector3(humanCarB.carBody.velocity.x, 0f, hum
anCarB.carBody.velocity.z).magnitude * 3.6f;
                if (carSpeedB > 999f) { carSpeedB = 999f; }
                carSpeedText[2].text = $"{(int)carSpeedB} km/h";
            }
            else
            { carSpeedText[0].text = $"{(int)carSpeed} km/h"; }
            yield return delay;
        }
```

```
    }
    public void ResumeRace()
    {
        IncreaseDepthOfField();
        StartCoroutine(StartCountdownTimer());
        StartCoroutine(FadeOut(pauseCanvas));
        StartCoroutine(FadeOut(buttonsCanvas));
        StartCoroutine(FadeOut(buttonsCanvas2P));
    }
    public void PauseRace()
    {
        DecreaseDepthOfField();
        FadeIn(pauseCanvas);
        StartCoroutine(FadeOut(buttonsCanvas));
        StartCoroutine(FadeOut(buttonsCanvas2P));
        if (raceOver)
        { StartCoroutine(FadeOut(overCanvasA)); StartCoroutine(FadeOut(results
CanvasA)); }
        else if (raceOverB)
        { StartCoroutine(FadeOut(overCanvasB)); StartCoroutine(FadeOut(results
CanvasB)); }
        Time.timeScale = 0f;
        racePaused = true;
    }
    private void DecreaseDepthOfField()
    {
        if (postProcessor.profile.TryGetSettings(out depthOfField))
        { LeanTween.value(6.5f, 1f, halfSecond).setEaseInOutQuart().setIgnoreT
imeScale(true).setOnUpdate((float depth) => { depthOfField.focusDistance.value
 = depth; }); }
    }
    private void IncreaseDepthOfField()
    {
        if (postProcessor.profile.TryGetSettings(out depthOfField))
        { LeanTween.value(1f, 6.5f, halfSecond).setEaseInOutQuart().setIgnoreT
imeScale(true).setOnUpdate((float depth) => { depthOfField.focusDistance.value
 = depth; }); }
    }
    public void IncreaseFieldOfView()
    {
        LeanTween.value(60f, 80f, 1f).setEaseInOutQuart().setIgnoreTimeScale(t
rue).setOnUpdate((float field) => { mainCamera.fieldOfView = field; });
    }
    public void DecreaseFieldOfView()
    {
        LeanTween.value(80f, 60f, 1f).setEaseInOutQuart().setIgnoreTimeScale(t
rue).setOnUpdate((float field) => { mainCamera.fieldOfView = field; });
    }
```

```
    public void IncreaseFieldOfViewB()
    {
        LeanTween.value(60f, 80f, 1f).setEaseInOutQuart().setIgnoreTimeScale(t
rue).setOnUpdate((float field) => { mainCameraB.fieldOfView = field; });
    }
    public void DecreaseFieldOfViewB()
    {
        LeanTween.value(80f, 60f, 1f).setEaseInOutQuart().setIgnoreTimeScale(t
rue).setOnUpdate((float field) => { mainCameraB.fieldOfView = field; });
    }
    public void ShowInfo(string infoString)
    {
        showingInfo = true;
        infoText.text = infoString;
        infoDisappearTime = Time.unscaledTime + 3f;
        infoCanvas.LeanAlpha(1f, halfSecond).setEaseInOutQuart().setIgnoreTime
Scale(true);
        if (delayedInfoAppear > 0f) { delayedInfoAppear -= 2f; }
    }
    public IEnumerator DelayedShowInfo(string infoString)
    {
        delayedInfoAppear += 2f;
        yield return new WaitForSecondsRealtime(delayedInfoAppear);
        ShowInfo(infoString);
    }
    public void ShowRaceOver()
    {
        cameraSwirl = true;
        humanCar.controlType = Player.ControlTypes.robot;
        FadeIn(overCanvas);
        StartCoroutine(FadeOut(countdownCanvas));
        StartCoroutine(FadeOut(raceCanvas));
        StartCoroutine(FadeOut(buttonsCanvas));
        StartCoroutine(FadeOut(timerCanvas));
        StartCoroutine(FadeOut(infoCanvas));
        StartCoroutine(ShowRaceResults());
        raceOver = true;
    }
    private IEnumerator ShowRaceResults()
    {
        animating = true;
        WaitForSecondsRealtime delay = new WaitForSecondsRealtime(0.1f);
        resultsText[0].text = carRank.ToString();
        if (humanCar.carEliminated)
        { resultsStars[0].SetActive(true); }
        else
        {   switch (humanCar.carRank)
            {
```

```
                case 1:
                    if ((gameType == Menu.GameTypes.play) && ((PlayerPrefs.Get
Int("Stars" + selectedLevel.ToString(), 0)) < 3))
                    { PlayerPrefs.SetInt("Stars" + selectedLevel.ToString(), 3
); }
                    resultsStars[3].SetActive(true);
                    break;
                case 2:
                    if ((gameType == Menu.GameTypes.play) && ((PlayerPrefs.Get
Int("Stars" + selectedLevel.ToString(), 0)) < 2))
                    { PlayerPrefs.SetInt("Stars" + selectedLevel.ToString(), 2
); }
                    resultsStars[2].SetActive(true);
                    break;
                case 3:
                    if ((gameType == Menu.GameTypes.play) && ((PlayerPrefs.Get
Int("Stars" + selectedLevel.ToString(), 0)) < 1))
                    { PlayerPrefs.SetInt("Stars" + selectedLevel.ToString(), 1
); }
                    resultsStars[1].SetActive(true);
                    break;
                default:
                    resultsStars[0].SetActive(true);
                    break;
            }
        }
        if (gameType == Menu.GameTypes.play)
        { resultsText[1].text = ("Level " + selectedLevel.ToString()); }
        else
        { resultsText[1].text = "1 Player"; }
        resultsText[2].text = mapName.ToString();
        resultsText[3].text = raceMode.ToString();
        resultsText[4].text = numLaps.ToString();
        resultsText[5].text = difficultyType.ToString();
        if ((gameType == Menu.GameTypes.play) && (resultsStars[3].activeSelf))
        {
            resultsText[20].text = string.Empty;
            switch (selectedLevel)
            {
                case 1:
                    if (PlayerPrefs.GetInt("Car2", 0) == 1) { break; }
                    PlayerPrefs.SetInt("Car2", 1);
                    resultsText[20].text = "NEW CAR UNLOCKED";
                    resultsText[21].text = GameManager.instance.cars[2].name;
                    break;
                case 2:
                    if (PlayerPrefs.GetInt("Level4", 0) == 1) { break; }
                    PlayerPrefs.SetInt("Level4", 1);
```

```
                    resultsText[20].text = "NEW LEVEL UNLOCKED";
                    resultsText[21].text = "LEVEL 4";
                    break;
                case 3:
                    if (PlayerPrefs.GetInt("Car3", 0) == 1) { break; }
                    PlayerPrefs.SetInt("Car3", 1);
                    resultsText[20].text = "NEW CAR UNLOCKED";
                    resultsText[21].text = GameManager.instance.cars[3].name;
                    break;
                case 4:
                    if (PlayerPrefs.GetInt("Level5", 0) == 1) { break; }
                    PlayerPrefs.SetInt("Level5", 1);
                    resultsText[20].text = "NEW LEVEL UNLOCKED";
                    resultsText[21].text = "LEVEL 5";
                    break;
                case 5:
                    if (PlayerPrefs.GetInt("Car4", 0) == 1) { break; }
                    PlayerPrefs.SetInt("Car4", 1);
                    resultsText[20].text = "NEW CAR UNLOCKED";
                    resultsText[21].text = GameManager.instance.cars[4].name;
                    break;
                default:
                    resultsText[20].text = string.Empty;
                    break;
            }
            if (resultsText[20].text != string.Empty)
            { unlockText.SetActive(true); }
        }
        yield return new WaitForSecondsRealtime(2f);
        UpdateLeaderboard(ref leaderboardName);
        FadeIn(resultsCanvas);
        animating = false;
        while (!(cars[cars.Count - 1].raceOver))
        { UpdateLeaderboard(ref leaderboardName); yield return delay; }
    }
    private void UpdateLeaderboard(ref TextMeshProUGUI name)
    {
        cars.Sort((carX, carY) => carX.carRank.CompareTo(carY.carRank));
        name.text = string.Empty;
        for (int i = 0; i < cars.Count; i++)
        {
            name.text += $"{cars[i].carRank}\t{cars[i].gameObject.name}";
            if ((gameType != Menu.GameTypes.quickRace2P) && (humanCar.gameObje
ct.GetInstanceID() == cars[i].gameObject.GetInstanceID()))
            { name.text += " (You)"; }
            else if ((gameType == Menu.GameTypes.quickRace2P) && (humanCar.gam
eObject.GetInstanceID() == cars[i].gameObject.GetInstanceID()))
            { name.text += " (A)"; }
```

```
            else if ((gameType == Menu.GameTypes.quickRace2P) && (humanCarB.ga
meObject.GetInstanceID() == cars[i].gameObject.GetInstanceID()))
            { name.text += " (B)"; }
            if (cars[i].carEliminated)
            { name.text += " (Out)"; }
            if (i != cars.Count - 1) { name.text += System.Environment.NewLine
; }
        }
    }
    private void UpdateLeaderboardTime(ref TextMeshProUGUI time)
    {
        time.text = string.Empty;
        for (int i = 0; i < cars.Count; i++)
        {
            if (cars[i].totalRaceTime > 599.99f) { cars[i].totalRaceTime = 599
.99f; }
            time.text += $"{(int)(cars[i].totalRaceTime)/60}:{(cars[i].totalRa
ceTime)%60:00.00}";
            if (i != cars.Count - 1) { time.text += System.Environment.NewLine
; }
        }
    }
    private IEnumerator RestartRace()
    {
        StartCoroutine(FadeOut(overCanvas));
        animating = true;
        FadeIn(loadingCanvas);
        yield return new WaitForSecondsRealtime(0.6f);
        PlayerPrefs.SetFloat("Volume", Menu.ins.soundVolume);
        SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex);
        animating = false;
    }
    private IEnumerator BackToMenu()
    {
        Menu.showLoadingCanvas = true;
        StartCoroutine(FadeOut(overCanvas));
        animating = true;
        FadeIn(loadingCanvas);
        Menu.ins.raceSong = music.audioSource.clip;
        PlayerPrefs.SetFloat("Volume", Menu.ins.soundVolume);
        PlayerPrefs.Save();
        yield return new WaitForSecondsRealtime(0.6f);
        SceneManager.LoadScene(0);
        animating = false;
    }
    private void FadeIn(CanvasGroup canvasGroup)
    {
        canvasGroup.alpha = 0f;
```

```
        canvasGroup.gameObject.SetActive(true);
        canvasGroup.LeanAlpha(1f, halfSecond).setEaseInOutQuart().setIgnoreTim
eScale(true);
    }
    private IEnumerator FadeOut(CanvasGroup canvasGroup)
    {
        animating = true;
        canvasGroup.LeanAlpha(0f, halfSecond).setEaseInOutQuart().setIgnoreTim
eScale(true);
        yield return halfSecondDelay;
        canvasGroup.gameObject.SetActive(false);
        canvasGroup.alpha = 1f;
        animating = false;
    }
}
```

5.2.3 - Script for Managing the Race (GameManager.cs)

This component is attached as a static class to the GameManager object, that is, each scene contains only one instance of the class. It first fetches all required attributes for the race, as decided by the player, from the static Menu class. It assigns the selected car to a random starting position, and then randomly selects other available cars as AI-controlled, and randomly sets their positions.

```
public class GameManager : MonoBehaviour
{
    private void Awake()
    {
        instance = this;
        inputController = GetComponentInChildren<InputController>();
        UIController = GetComponentInChildren<UIController>();
        driftController = GetComponentInChildren<DriftController>();
        gameType = Menu.ins.gameType;
        raceMode = Menu.ins.raceMode;
        mapName = Menu.ins.mapName;
        difficultyType = Menu.ins.difficultyType;
        numLaps = Menu.ins.numLaps;
        numCars = Menu.ins.numCars;
        selectedLevel = Menu.ins.selectedLevel;
        selectedCar = Menu.ins.selectedCar;
        if (gameType == Menu.GameTypes.quickRace2P)
        {
            inputControllerB = GetComponentInChildren<InputControllerB>();
            UIController.mainCamera.gameObject.SetActive(true);
            UIController.mainCamera.rect = new Rect(0.5f, 0f, 0.5f, 1f);
            UIController.mainCameraB.gameObject.SetActive(true);
            UIController.mainCameraB.rect = new Rect(0f, 0f, 0.5f, 1f);
            selectedCarB = Menu.ins.selectedCarB;
        }
```

```
        else
        {
            UIController.mainCamera.gameObject.SetActive(true);
            UIController.mainCamera.rect = new Rect(0f, 0f, 1f, 1f);
        }
        totalNumCars = Menu.ins.totalNumCars;
        randomNum = new System.Random();
        cars = new List<GameObject>();
        startingPositions = new List<Transform>();
        startingPositionsSubParent = (startingPositionsParent.transform.GetChi
ld(numCars - 1)).gameObject;
        foreach (Transform car in carsParent.transform)
        { cars.Add(car.gameObject); }
        foreach (Transform position in startingPositionsSubParent.transform)
        { startingPositions.Add(position); }
        for (int i = 0; i < totalNumCars; i++)
        { cars[i].SetActive(false); }
        AudioListener.volume = volumeSlider.value = Menu.ins.soundVolume;
        AssignCarPositions();
    }
    private void AssignCarPositions()
    {
        selectedCarPosition = randomNum.Next(0, numCars);
        cars[selectedCar].transform.position = startingPositions[selectedCarPo
sition].transform.position;
        cars[selectedCar].SetActive(true);
        cars[selectedCar].GetComponent<Player>().controlType = Player.ControlT
ypes.human;
        humanCar = cars[selectedCar].GetComponent<Transform>();
        if (gameType == Menu.GameTypes.quickRace2P)
        {
            selectedCarPositionB = randomNum.Next(0, numCars - 1);
            if (selectedCarPositionB >= selectedCarPosition)
            { selectedCarPositionB++; }
            cars[selectedCarB].transform.position = startingPositions[selected
CarPositionB].transform.position;
            cars[selectedCarB].SetActive(true);
            cars[selectedCarB].GetComponent<Player>().controlType = Player.Con
trolTypes.humanB;
            humanCarB = cars[selectedCarB].GetComponent<Transform>();
        }
        for (int i = 0; i < numCars; i++)
        {
            if ((i == selectedCarPosition) || ((i == selectedCarPositionB) &&
(gameType == Menu.GameTypes.quickRace2P)))
            { continue; }
            carPositionAssigned = false;
            while (!carPositionAssigned)
```

```
            {
                randomCarIndex = randomNum.Next(0, totalNumCars);
                if (cars[randomCarIndex].activeSelf) { continue; }
                cars[randomCarIndex].transform.position = startingPositions[i]
.transform.position;
                cars[randomCarIndex].SetActive(true);
                cars[randomCarIndex].GetComponent<Player>().controlType = Play
er.ControlTypes.robot;
                carPositionAssigned = true;
            }
        }
    }
}
```

5.2.4 - Script for controlling the Car's Physical Attributes (CarController.cs)

This component is attached to each car. It initialises attributes such the maximum throttle of the engine, the maximum steering ability, and the car's centre of mass. Every frame, it applies a torque and an angle to each wheel of the car.

```
public class CarController : MonoBehaviour
{
    private void Start()
    {
        maxThrottle = 2000f;
        maxSteer = 15f;
        wheels = GetComponentsInChildren<Wheel>();
        carBody = GetComponent<Rigidbody>();
        carBody.centerOfMass = centerOfMass.localPosition;
    }
    private void Update()
    {
        foreach (Wheel wheel in wheels)
        {
            wheel.torque = throttle * maxThrottle;
            wheel.steerAngle = steer * maxSteer;
        }
    }
}
```

5.2.5 - Script for Controlling Input of Player A (InputController.cs)

This component is attached as a child of the static GameManager class. It defines and returns the value and their associated actions of the virtual axes defined in Unity's Input Manager. Player A is controlled by "Horizontal" and "Vertical" axes (arrow keys).

```
public class InputController : MonoBehaviour
{
    private void Awake()
    { inputSteerAxis = "Horizontal"; inputThrottleAxis = "Vertical"; }
    private void Update()
    {
        steerInput = Input.GetAxis(inputSteerAxis);
        throttleInput = Input.GetAxis(inputThrottleAxis);
    }
}
```

5.2.6 - Script for Controlling Input of Player B (InputControllerB.cs)

This component is similar to the InputController class. It controls the input of Player B in 2-player mode. Player B is controlled by "HorizontalB" and "VerticalB" axes (WASD keys).

```
public class InputControllerB : MonoBehaviour
{
    private void Awake()
    { inputSteerAxisB = "HorizontalB"; inputThrottleAxisB = "VerticalB"; }
    private void Update()
    {
        steerInputB = Input.GetAxis(inputSteerAxisB);
        throttleInputB = Input.GetAxis(inputThrottleAxisB);
    }
}
```

5.2.7 - Script for the Camera to Follow the Car (CameraFollow.cs)

This component is attached to each camera in every scene. It fetches the human-controlled car from the GameManager, and then moves the camera according to the car's position.

```
public class CameraFollow : MonoBehaviour
{
    private void LateUpdate()
    {
        if (GameManager.instance.UIController.cameraSwirl)
        { inFrontCar = false; BesideCar(); }
        else if ((inFrontCar) && (!GameManager.instance.UIController.cameraSwi
rl))
        { InFrontCar(); }
        else
        { BehindCar(); }
        MoveCamera();
    }
    private void BesideCar()
    {
```

```
        wantedRotationAngle = humanCar.eulerAngles.y - 150f;
        wantedHeight = Mathf.Abs(humanCar.position.y) * (height - 1f);
        if (wantedHeight > 100f) { wantedHeight = 100f; }
    }
    private void BehindCar()
    {
        wantedRotationAngle = humanCar.eulerAngles.y;
        wantedHeight = Mathf.Abs(humanCar.position.y) * height;
        if (wantedHeight > 750f) { wantedHeight = 750f; }
    }
    private void InFrontCar()
    {
        wantedRotationAngle = humanCar.eulerAngles.y;
        wantedHeight = Mathf.Abs(humanCar.position.y) * (height - 4f);
        if (wantedHeight > 750f) { wantedHeight = 750f; }
    }
    private void MoveCamera()
    {
        if (inFrontCar)
        { currentRotationAngle = Mathf.LerpAngle(currentRotationAngle, wantedR
otationAngle, 3f * rotationDamping * Time.unscaledDeltaTime); }
        else
        { currentRotationAngle = Mathf.LerpAngle(currentRotationAngle, wantedR
otationAngle, rotationDamping * Time.unscaledDeltaTime); }
        currentHeight = Mathf.Lerp(currentHeight, wantedHeight, heightDamping
* Time.unscaledDeltaTime);
        currentRotation = Quaternion.Euler(0f, currentRotationAngle, 0f);
        transform.position = humanCar.position;
        transform.position -= currentRotation * Vector3.forward * distance;
        transform.position = new Vector3 (transform.position.x, currentHeight,
 transform.position.z);
        transform.LookAt(humanCar);
        if (inFrontCar)
        { transform.position += currentRotation * Vector3.forward * (distance
+ 4f); }
    }
    private IEnumerator ChangeCameraView()
    {
        if (animating) { yield break; }
        animating = true;
        inFrontCar = !inFrontCar;
        yield return new WaitForSecondsRealtime(1f);
        animating = false;
    }
}
```

5.2.8 - Script for Controlling Each Wheel (Wheel.cs)

This component is attached to every wheel of every car. It controls whether the wheel if steerable (e.g front wheels) and is powered by the engine (e.g 4-wheel drive). It also updates the drift marks left by each wheel by accessing the static DriftController class (imported asset).

```
public class Wheel : MonoBehaviour
{
    private void Awake()
    {
        minDriftSpeed = 0.5f;
        maxDriftIntensity = 10f;
        wheelSlipMultiplier = 1000f;
        lastDriftIndex = -1;
    }
    private void Update()
    {
        wheelCollider.GetWorldPose(out wheelPosition, out wheelRotation);
        wheelTransform.position = wheelPosition;
        wheelTransform.rotation = wheelRotation;
    }
    private void FixedUpdate()
    {
        if (power)
        { wheelCollider.motorTorque = torque; }
        if (steer)
        { wheelCollider.steerAngle = steerAngle * (inverseSteer ? -1 : 1); }
        lastFixedUpdateTime = Time.time;
    }
    private void LateUpdate()
    {
        if (wheelCollider.GetGroundHit(out wheelHitInfo))
        {
            GetWheelInfo();
            if (totalDrift >= minDriftSpeed)
            {
                driftIntensity = Mathf.Clamp((totalDrift / maxDriftIntensity),
 0, 1);
                driftPoint = wheelHitInfo.point + transform.forward + (carBody
.velocity * (Time.time - lastFixedUpdateTime));
                lastDriftIndex = GameManager.instance.driftController.AddDrift
Mark(driftPoint, wheelHitInfo.normal, driftIntensity, lastDriftIndex);
                if ((driftIntensity > 0.5f) && (Mathf.Abs(carForwardVelocity)
> 2f))
                {
                    smokeEmission.rateOverTime = (driftIntensity * 100f) - 50f
;
                    if (!driftSound.isPlaying) { driftSound.Play();}
```

```
                }
                driftSound.volume = driftIntensity;
            }
            else { NoDrift(); }
        }
        else { NoDrift(); }
    }
    private void GetWheelInfo()
    {
        localVelocity = transform.InverseTransformDirection(carBody.velocity);
        totalDrift = Mathf.Abs(localVelocity.x);
        wheelAngularVelocity = wheelCollider.radius * ((2f * Mathf.PI * wheelC
ollider.rpm) / 60f);
        carForwardVelocity = Vector3.Dot(carBody.velocity, transform.forward);
        wheelSpin = Mathf.Abs(carForwardVelocity - wheelAngularVelocity) * whe
elSlipMultiplier;
        wheelSpin = Mathf.Max(0f, wheelSpin * (5f - Mathf.Abs(carForwardVeloci
ty)));
        totalDrift += wheelSpin;
    }
    private void NoDrift()
    { smokeEmission.rateOverTime = 0f; lastDriftIndex = -1; }
}
```

5.2.9 - Script for Controlling the Main Menu (Menu.cs)

This component is a static class in the level 0 scene. It controls the activation and deactivation of canvases (level selection, car selection, etc) of the entire main menu. It also sets the race attributes as selected by the player, which is be used by the GameManager class when the level is loaded.

```
public class Menu : MonoBehaviour
{
    private void LateUpdate()
    {
        if (menuScreen == MenuScreens.cars) { cars[selectedCar].Rotate(0f, 10f
 * Time.deltaTime, 0f); }
        else if (menuScreen == MenuScreens.carsB) { carsB[selectedCarB].Rotate
(0f, 10f * Time.deltaTime, 0f); }
        if (animating) { return; }
        switch (menuScreen)
        {
            case MenuScreens.levels: LevelMenuKeyPress(); break;
            case MenuScreens.settings: RaceSettingsMenuKeyPress(); break;
            case MenuScreens.cars: CarMenuKeyPress(); break;
            case MenuScreens.carsB: CarBMenuKeyPress(); break;
            default: MainMenuKeyPress(); break;
        }
    }
```

```
    private IEnumerator LaunchGame()
    {
        animating = true;
        if (showLoadingCanvas)
        {
            loadingCanvas.alpha = 1f;
            loadingCanvas.gameObject.SetActive(true);
        }
        soundVolume = PlayerPrefs.GetFloat("Volume", 2f);
        if ((soundVolume < 0f) || (soundVolume > 1f))
        { soundVolume = 1f; }
        AudioListener.volume = volumeSlider.value = soundVolume;
        cameraAnimation.Play("MenuCamera", -1, 0f);
        totalNumLevels = levelsLayout.gameObject.transform.childCount;
        halfSecond = 0.5f;
        halfSecondDelay = new WaitForSecondsRealtime(halfSecond);
        windowOpen = false;
        carsInstantiated = false;
        carsInstantiatedB = false;
        selectedCar = 0;
        launchCanvas.alpha = 1f;
        launchCanvas.gameObject.SetActive(true);
        launchCanvas.LeanAlpha(0f, 2f).setEaseInOutQuart().setIgnoreTimeScale(
true);
        loadingCanvas.LeanAlpha(0f, 1f).setEaseInOutQuart().setIgnoreTimeScale
(true);
        animating = true;
        yield return new WaitForSecondsRealtime(2f);
        launchCanvas.gameObject.SetActive(false);
        launchCanvas.alpha = 1f;
        loadingCanvas.gameObject.SetActive(false);
        loadingCanvas.alpha = 1f;
        animating = false;
    }
    private IEnumerator StartRace()
    {
        StartCoroutine(FadeOut(carSelectionCanvas));
        StartCoroutine(FadeOut(carSelectionCanvasB));
        animating = true;
        FadeIn(loadingCanvas);
        Mathf.Clamp(selectedCar, 0, (totalNumCars - 1));
        PlayerPrefs.SetFloat("Volume", soundVolume);
        PlayerPrefs.Save();
        menuSong = music.audioSource.clip;
        music.audioSource.Stop();
        yield return new WaitForSecondsRealtime(2f);
        SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex + sele
ctedLevel);
```

```
        animating = false;
    }
    public void SetRaceSettings()
    {
        if (menuScreen == MenuScreens.levels)
        {
            selectedLevel = int.Parse(levelsLayout.ActiveToggles().FirstOrDefa
ult().gameObject.name);
            Mathf.Clamp(selectedLevel, 1, totalNumLevels);
            switch (selectedLevel)
            {
                case 2: mapName = MapNames.Artic; raceMode = RaceModes.Escape;
 break;
                case 3: mapName = MapNames.Desert; raceMode = RaceModes.Surviv
al; break;
                case 4: mapName = MapNames.Urban; raceMode = RaceModes.Infecti
on; break;
                case 5: mapName = MapNames.Moon; raceMode = RaceModes.Special;
 break;
                default: mapName = MapNames.Rural; raceMode = RaceModes.Classi
c; break;
            }
        }
        switch (gameType)
        {
            case GameTypes.play:
                difficultyType = DifficultyTypes.Normal;
                numLaps = 3;
                numCars = 4;
                break;
            default:
                difficultyType = (DifficultyTypes) int.Parse(difficultiesLayou
t.ActiveToggles().FirstOrDefault().gameObject.name);
                raceMode = (RaceModes) int.Parse(modesLayout.ActiveToggles().F
irstOrDefault().gameObject.name);
                numLaps = int.Parse(numLapsLayout.ActiveToggles().FirstOrDefau
lt().gameObject.name);
                Mathf.Clamp(numLaps, 1, 10);
                numCars = int.Parse(numCarsLayout.ActiveToggles().FirstOrDefau
lt().gameObject.name);
                if (gameType == GameTypes.quickRace1P)
                { Mathf.Clamp(numCars, 2, totalNumCars); }
                else
                { Mathf.Clamp(numCars, 1, totalNumCars); }
                break;
        }
    }
    private void MainMenuKeyPress()
```

```
    {
        if (Input.GetKeyDown(KeyCode.Return) && (!windowOpen))
        { ClickPlay(); music.PlayClickSound(); }
        else if (Input.GetKeyDown(KeyCode.Escape) && (!windowOpen))
        { OpenWindow(exitCanvas); music.PlayClickSound(); }
        else if (Input.GetKeyDown(KeyCode.Return) && (exitCanvas.gameObject.ac
tiveSelf))
        { ExitGame(); music.PlayClickSound(); }
    }
    private void LevelMenuKeyPress()
    {
        if ((Input.GetKeyDown(KeyCode.Return)) && (gameType == GameTypes.play)
)
        { SelectCar(); music.PlayClickSound(); }
        else if ((Input.GetKeyDown(KeyCode.Return)) && (gameType != GameTypes.
play))
        { SetQuickRaceSettings(); music.PlayClickSound(); }
        else if (Input.GetKeyDown(KeyCode.Escape))
        { Start(); music.PlayClickSound(); }
    }
    private void RaceSettingsMenuKeyPress()
    {
        if (Input.GetKeyDown(KeyCode.Return))
        { SelectCar(); music.PlayClickSound(); }
        else if ((Input.GetKeyDown(KeyCode.Escape)) && (gameType == GameTypes.
quickRace1P))
        { ClickQuickRace1P(); music.PlayClickSound(); }
        else if ((Input.GetKeyDown(KeyCode.Escape)) && (gameType == GameTypes.
quickRace2P))
        { ClickQuickRace2P(); music.PlayClickSound(); }
    }
    private void CarMenuKeyPress()
    {
        if ((Input.GetKeyDown(KeyCode.Return)) && (gameType != GameTypes.quick
Race2P) && (!carLockText.gameObject.activeInHierarchy))
        { StartRaceWrapper(); music.PlayStartSound(); }
        else if ((Input.GetKeyDown(KeyCode.Return)) && (gameType == GameTypes.
quickRace2P))
        { SelectCarB(); music.PlayClickSound(); }
        else if ((Input.GetKeyDown(KeyCode.Escape)) && (gameType == GameTypes.
play))
        { ClickPlay(); music.PlayClickSound(); }
        else if ((Input.GetKeyDown(KeyCode.Escape)) && (gameType != GameTypes.
play))
        { SetQuickRaceSettings(); music.PlayClickSound(); }
        else if (Input.GetKeyDown(KeyCode.LeftArrow) && (selectedCar != 0))
        { PreviousCar(); music.PlayClickSound(); }
```

```
        else if (Input.GetKeyDown(KeyCode.RightArrow) && (selectedCar != total
NumCars - 1))
        { NextCar(); music.PlayClickSound(); }
    }
    private void CarBMenuKeyPress()
    {
        if ((Input.GetKeyDown(KeyCode.Return)) && (!carLockTextB.gameObject.ac
tiveInHierarchy))
        { StartRaceWrapper(); music.PlayStartSound(); }
        else if (Input.GetKeyDown(KeyCode.Escape))
        { SelectCar(); music.PlayClickSound(); }
        else if (Input.GetKeyDown(KeyCode.A) && (selectedCarB != 0))
        { PreviousCarB(); music.PlayClickSound(); }
        else if (Input.GetKeyDown(KeyCode.D) && (selectedCarB != totalNumCars
- 1))
        { NextCarB(); music.PlayClickSound(); }
    }
    private IEnumerator OpenResetGameWindow()
    {
        Menu.ins.animating = true;
        StartCoroutine(Menu.ins.FadeOut(optionsCanvas));
        yield return new WaitForSecondsRealtime(0.2f);
        Menu.ins.FadeIn(resetGameCanvas);
    }
    private IEnumerator CloseResetGameWindow()
    {
        Menu.ins.animating = true;
        StartCoroutine(Menu.ins.FadeOut(resetGameCanvas));
        yield return new WaitForSecondsRealtime(0.2f);
        Menu.ins.FadeIn(optionsCanvas);
    }
    private IEnumerator ResetGame()
    {
        animating = true;
        Menu.showLoadingCanvas = true;
        FadeIn(loadingCanvas);
        yield return new WaitForSecondsRealtime(0.6f);
        PlayerPrefs.DeleteAll();
        SceneManager.LoadScene(0);
        animating = false;
    }
    public void FadeIn(CanvasGroup canvasGroup)
    {
        canvasGroup.alpha = 0f;
        canvasGroup.gameObject.SetActive(true);
        canvasGroup.LeanAlpha(1f, halfSecond).setEaseInOutQuart().setIgnoreTim
eScale(true);
    }
```

```
    public IEnumerator FadeOut(CanvasGroup canvasGroup)
    {
        animating = true;
        canvasGroup.LeanAlpha(0f, halfSecond).setEaseInOutQuart().setIgnoreTim
eScale(true);
        yield return halfSecondDelay;
        canvasGroup.gameObject.SetActive(false);
        canvasGroup.alpha = 1f;
        animating = false;
    }
}
```

5.2.10 - Script for Drawing the Road Path (TrackWaypoints.cs)

This component is attached to an object and has several invisible children which are placed along the road. The Gizmos class built in Unity is used to join each child into a path. This line is used by the AI of the robot cars and by the real-time ranking system implemented in each car.

```
public class TrackWaypoints : MonoBehaviour
{
    private void OnDrawGizmos()
    {
        Gizmos.color = Color.white;
        path = GetComponentsInChildren<Transform>();
        waypoints = new List<Transform>();
        for (int i = 1; i < path.Length; i++)
        { waypoints.Add(path[i]); }
        for (int i = 0; i < waypoints.Count; i++)
        {
            currentWaypoint = waypoints[i].position;
            previousWaypoint = Vector3.zero;
            if (i != 0)
            { previousWaypoint = waypoints[i - 1].position; }
            else if (i == 0)
            { previousWaypoint = waypoints[waypoints.Count - 1].position; }
            Gizmos.DrawLine(previousWaypoint, currentWaypoint);
            Gizmos.DrawSphere(currentWaypoint, 1f);
        }
    }
}
```

5.2.11 - Script for Engine Sound (EngineSound.cs)

This component is attached to each car. It varies the pitch of the sound of the engine according to the speed of the car.

```
public class EngineSound : MonoBehaviour
{
    void Start()
    { engineSound = GetComponent<AudioSource>(); }
    void LateUpdate()
    { engineSound.pitch = carBody.velocity.magnitude / 50f; }
}
```

5.2.12 - Script for Game Music (Music.cs)

This component is attached to each camera (cameras have an audio listener) of every scene. It randomises an array of songs and plays them one after the other. The player can choose to change to the next or previous song as they wish.

```
public class Music : MonoBehaviour
{
    private void Shuffle()
    {
        AudioClip temp;
        int i, j;
        System.Random randomNum;
        i = songs.Length;
        j = 0;
        randomNum = new System.Random();
        while (i > 1)
        {
            i--;
            j = randomNum.Next(0, i + 1);
            temp = songs[j];
            songs[j] = songs[i];
            songs[i] = temp;
        }
    }
    private void Update()
    { if (!audioSource.isPlaying) { NextSong(); } }
    public void NextSong()
    {
        if (songIndex >= songs.Length - 1)
        { songIndex = 0; }
        else
        { songIndex++; }
        if ((Menu.ins.menuSong == songs[songIndex]) || (Menu.ins.raceSong == s
ongs[songIndex]))
        { NextSong(); }
        PlaySong();
    }
    public void PreviousSong()
    {
        if (songIndex <= 0)
```

```
        { songIndex = songs.Length - 1; }
        else
        { songIndex--; }
        PlaySong();
    }
    private void PlaySong()
    {
        audioSource.clip = songs[songIndex];
        audioSource.Play();
        songString = audioSource.clip.ToString();
        songName.text = (songString).Substring(0, songString.Length - 24);
    }
    public void ChangeVolume(float sliderValue)
    { AudioListener.volume = Menu.ins.soundVolume = sliderValue; }
}
```

5.2.13 - Script for Level Selection Screen (LevelSelection.cs)

This component is attached as a child of the static Menu class. It checks the previous menu screen and activates buttons accordingly. It also locks or unlocks levels according to values stored in the Windows Registry.

```
public class LevelSelection : MonoBehaviour
{
    public void SelectLevelWrapper() { StartCoroutine(SelectLevel()); }

    private IEnumerator SelectLevel()
    {
        if (Menu.ins.gameType == Menu.GameTypes.play)
        {
            Menu.ins.nextButtonLevelPlay.gameObject.SetActive(true);
            Menu.ins.nextButtonLevelQuickRace.gameObject.SetActive(false);
        }
        else
        {
            Menu.ins.nextButtonLevelPlay.gameObject.SetActive(false);
            Menu.ins.nextButtonLevelQuickRace.gameObject.SetActive(true);
        }
        Menu.ins.FadeIn(Menu.ins.levelSelectionCanvas);
        Menu.ins.carsParent.SetActive(false);
        if (Menu.ins.menuScreen == Menu.MenuScreens.main)
        {
            Menu.ins.menuScreen = Menu.MenuScreens.levels;
            Menu.ins.mainCars.SetActive(false);
            StartCoroutine(Menu.ins.FadeOut(Menu.ins.mainMenuCanvas));
        }
        else if (Menu.ins.menuScreen == Menu.MenuScreens.settings)
        {
            Menu.ins.menuScreen = Menu.MenuScreens.levels;
```

```
            StartCoroutine(Menu.ins.FadeOut(Menu.ins.raceSettingsCanvas));
            Menu.ins.cameraAnimation.enabled = true;
        }
        else if (Menu.ins.menuScreen == Menu.MenuScreens.cars)
        {
            Menu.ins.menuScreen = Menu.MenuScreens.levels;
            StartCoroutine(Menu.ins.FadeOut(Menu.ins.carSelectionCanvas));
            Menu.ins.mainCamera.LeanMove(Menu.ins.cameraPosition, Menu.ins.hal
fSecond).setEaseInOutQuart().setIgnoreTimeScale(true);
            Menu.ins.mainCamera.LeanRotate(Menu.ins.cameraRotation, Menu.ins.h
alfSecond).setEaseInOutQuart().setIgnoreTimeScale(true);
            yield return new WaitForSecondsRealtime(0.5f);
            Menu.ins.cameraAnimation.enabled = true;
        }
        for (int i = 0; i < Menu.ins.totalNumLevels; i++)
        {
            level = Menu.ins.levelsLayout.gameObject.transform.GetChild(i);
            levelStars = level.GetChild(4);
            levelLock = level.GetChild(5);
            levelMode = level.GetChild(6);
            levelToggle = level.GetComponent<Toggle>();
            if (Menu.ins.gameType == Menu.GameTypes.play)
            {
                if (i <= 2) { PlayerPrefs.SetInt("Level" + (i + 1).ToString(),
 1); }
                if (PlayerPrefs.GetInt("Level" + (i + 1).ToString(), 0) == 0)
                { CheckSelection(); LockLevel(); }
                else
                { numStars = PlayerPrefs.GetInt("Stars" + (i + 1).ToString(),
0); UnlockLevel(); }
            }
            else
            { QuickRaceLevel(); }
        }
    }
    private void LockLevel()
    {
        levelStars.gameObject.SetActive(false);
        levelLock.gameObject.SetActive(true);
        levelMode.gameObject.SetActive(true);
        levelToggle.interactable = false;
    }
    private void UnlockLevel()
    {
        SetStars();
        levelStars.gameObject.SetActive(true);
        levelLock.gameObject.SetActive(false);
        levelMode.gameObject.SetActive(true);
```

```
        levelToggle.interactable = true;
    }
    private void SetStars()
    {
        for (int i = 0; i <= 2; i++)
        { levelStars.GetChild(i).gameObject.SetActive(false); }
        if ((numStars >= 1) && (numStars <= 3))
        { levelStars.GetChild(numStars - 1).gameObject.SetActive(true); }
    }
    private void QuickRaceLevel()
    {
        levelStars.gameObject.SetActive(false);
        levelLock.gameObject.SetActive(false);
        levelMode.gameObject.SetActive(false);
        levelToggle.interactable = true;
    }
    private void CheckSelection()
    {
        if (!levelToggle.isOn) { return; }
        Menu.ins.levelsLayout.transform.GetChild(0).GetComponent<Toggle>().isO
n = true;
        Menu.ins.levelScrollbar.value = 0;
    }
}
```

5.2.14 - Script for Car Selection Screen (CarSelection.cs)

This component is attached as a child of the static Menu class. Similarly to the LevelSelection class, it checks the previous menu screen and activates buttons accordingly. It also locks or unlocks car according to values stored in the Windows Registry. It controls the car which is currently being shown and moves them out of the screen as decided by the player.

```
public class CarSelection : MonoBehaviour
{
    public void SelectCar()
    {
        if (Menu.ins.gameType == Menu.GameTypes.play)
        {
            Menu.ins.playButtonCar.gameObject.SetActive(true);
            Menu.ins.nextButtonCar.gameObject.SetActive(false);
            Menu.ins.backButtonCarPlay.gameObject.SetActive(true);
            Menu.ins.backButtonCarQuickRace.gameObject.SetActive(false);
            Menu.ins.carSelectionTextA.SetActive(false);
            Menu.ins.carSelectionImageA.SetActive(false);
        }
        else if (Menu.ins.gameType == Menu.GameTypes.quickRace1P)
        {
            Menu.ins.playButtonCar.gameObject.SetActive(true);
```

```
            Menu.ins.nextButtonCar.gameObject.SetActive(false);
            Menu.ins.backButtonCarPlay.gameObject.SetActive(false);
            Menu.ins.backButtonCarQuickRace.gameObject.SetActive(true);
            Menu.ins.carSelectionTextA.SetActive(false);
            Menu.ins.carSelectionImageA.SetActive(false);
        }
        else if (Menu.ins.gameType == Menu.GameTypes.quickRace2P)
        {
            Menu.ins.playButtonCar.gameObject.SetActive(false);
            Menu.ins.nextButtonCar.gameObject.SetActive(true);
            Menu.ins.backButtonCarPlay.gameObject.SetActive(false);
            Menu.ins.backButtonCarQuickRace.gameObject.SetActive(true);
            Menu.ins.carSelectionTextA.SetActive(true);
            Menu.ins.carSelectionImageA.transform.localScale = new Vector3(1f,
 1f, 1f);
            Menu.ins.carSelectionImageA.SetActive(true);
            Menu.ins.carSelectionImageA.LeanScale(new Vector3(0f, 0f, 0f), 1.5
f).setEaseInOutQuart().setIgnoreTimeScale(true);;
        }
        Menu.ins.cameraAnimation.enabled = false;
        if (Menu.ins.menuScreen != Menu.MenuScreens.carsB)
        {
            Menu.ins.cameraPosition = Menu.ins.mainCamera.position;
            Menu.ins.cameraRotation = Menu.ins.mainCamera.eulerAngles;
        }
        Menu.ins.FadeIn(Menu.ins.carSelectionCanvas);
        if (Menu.ins.menuScreen == Menu.MenuScreens.levels)
        {
            Menu.ins.SetRaceSettings();
            Menu.ins.menuScreen = Menu.MenuScreens.cars;
            StartCoroutine(Menu.ins.FadeOut(Menu.ins.levelSelectionCanvas));
        }
        else if (Menu.ins.menuScreen == Menu.MenuScreens.settings)
        {
            Menu.ins.SetRaceSettings();
            Menu.ins.menuScreen = Menu.MenuScreens.cars;
            StartCoroutine(Menu.ins.FadeOut(Menu.ins.raceSettingsCanvas));
        }
        else if (Menu.ins.menuScreen == Menu.MenuScreens.carsB)
        {
            Menu.ins.menuScreen = Menu.MenuScreens.cars;
            StartCoroutine(Menu.ins.FadeOut(Menu.ins.carSelectionCanvasB));
        }
        Menu.ins.mainCamera.LeanMove(new Vector3(7f, 2.5f, 2f), Menu.ins.halfS
econd).setEaseInOutQuart().setIgnoreTimeScale(true);
        Menu.ins.mainCamera.LeanRotate(new Vector3(15f, 253f, 0f), Menu.ins.ha
lfSecond).setEaseInOutQuart().setIgnoreTimeScale(true);
        if (Menu.ins.cars.Count <= 0)
```

```
        {
            foreach (Transform car in Menu.ins.carsParent.transform)
            { Menu.ins.cars.Add(car); }
        }
        if (Menu.ins.carsInstantiatedB)
        { Menu.ins.cars[Menu.ins.selectedCar].rotation = Menu.ins.carsB[Menu.i
ns.selectedCarB].rotation; }
        else
        { Menu.ins.cars[Menu.ins.selectedCar].rotation = Quaternion.Euler(0f,
-90f, 0f); }
        Menu.ins.carsParentB.SetActive(false);
        Menu.ins.carsParent.SetActive(true);
        if (Menu.ins.selectedCar != 0) { CheckCarLock(); }
        if (Menu.ins.carsInstantiated) { return; }
        Menu.ins.previousCarPosition = -30f;
        Menu.ins.currentCarPosition = 0f;
        Menu.ins.nextCarPosition = 30f;
        tenthSecondDelay = new WaitForSecondsRealtime(0.1f);
        quarterSecondDelay = new WaitForSecondsRealtime(0.25f);
        twoFifthsSecondDelay = new WaitForSecondsRealtime(0.4f);
        carLockObject = (Menu.ins.carLockText.gameObject.transform.parent).gam
eObject;
        carLockImage = carLockObject.GetComponent<Image>();
        carLockObject.transform.localScale = new Vector3(0f, 0f, 0f);
        Menu.ins.selectedCar = 0;
        Menu.ins.totalNumCars = Menu.ins.cars.Count;
        for (int i = 0; i < Menu.ins.totalNumCars; i ++)
        { Menu.ins.cars[i].position = new Vector3(Menu.ins.cars[i].position.x,
 Menu.ins.cars[i].position.y, Menu.ins.nextCarPosition); }
        Menu.ins.cars[Menu.ins.selectedCar].position = new Vector3(Menu.ins.ca
rs[Menu.ins.selectedCar].position.x, Menu.ins.cars[Menu.ins.selectedCar].posit
ion.y, Menu.ins.currentCarPosition);
        Menu.ins.carName.text = Menu.ins.cars[Menu.ins.selectedCar].gameObject
.name;
        Menu.ins.previousCarButton.interactable = false;
        if (Menu.ins.totalNumCars > 1)
        { Menu.ins.nextCarButton.interactable = true; }
        else
        { Menu.ins.nextCarButton.interactable = false; }
        for (int i = 0; i <= 1; i++)
        { Menu.ins.carLockTexts[i] = string.Empty; }
        Menu.ins.carLockTexts[2] = "Get 3 stars in \n Level 1 to unlock";
        Menu.ins.carLockTexts[3] = "Get 3 stars in \n Level 3 to unlock";
        Menu.ins.carLockTexts[4] = "Get 3 stars in \n Level 5 to unlock";
        CheckCarLock();
        Menu.ins.carsInstantiated = true;
    }
    public IEnumerator NextCar()
```

```
    {
        Menu.ins.animating = true;
        Menu.ins.cars[Menu.ins.selectedCar].LeanMoveZ(Menu.ins.previousCarPosi
tion, Menu.ins.halfSecond).setEaseInOutQuart().setIgnoreTimeScale(true);
        Menu.ins.selectedCar++;
        Menu.ins.cars[Menu.ins.selectedCar].rotation = Quaternion.Euler(0f, -
90f, 0f);
        Menu.ins.cars[Menu.ins.selectedCar].LeanMoveZ(Menu.ins.currentCarPosit
ion, Menu.ins.halfSecond).setEaseInOutQuart().setIgnoreTimeScale(true);
        Menu.ins.previousCarButton.interactable = true;
        if (Menu.ins.selectedCar == Menu.ins.totalNumCars - 1)
        { Menu.ins.nextCarButton.interactable = false; }
        CheckCarLock();
        Menu.ins.carName.gameObject.LeanScaleX(0f, 0.1f).setEaseInOutQuart().s
etIgnoreTimeScale(true);;
        yield return tenthSecondDelay;
        Menu.ins.carName.text = Menu.ins.cars[Menu.ins.selectedCar].gameObject
.name;
        Menu.ins.carName.gameObject.LeanScaleX(1f, 0.1f).setEaseInOutQuart().s
etIgnoreTimeScale(true);;
        yield return twoFifthsSecondDelay;
        Menu.ins.animating = false;
    }
    public IEnumerator PreviousCar()
    {
        Menu.ins.animating = true;
        Menu.ins.cars[Menu.ins.selectedCar].LeanMoveZ(Menu.ins.nextCarPosition
, Menu.ins.halfSecond).setEaseInOutQuart().setIgnoreTimeScale(true);
        Menu.ins.selectedCar--;
        Menu.ins.cars[Menu.ins.selectedCar].rotation = Quaternion.Euler(0f, -
90f, 0f);
        Menu.ins.cars[Menu.ins.selectedCar].LeanMoveZ(Menu.ins.currentCarPosit
ion, Menu.ins.halfSecond).setEaseInOutQuart().setIgnoreTimeScale(true);
        Menu.ins.nextCarButton.interactable = true;
        if (Menu.ins.selectedCar == 0)
        { Menu.ins.previousCarButton.interactable = false; }
        CheckCarLock();
        Menu.ins.carName.gameObject.LeanScaleX(0f, 0.1f).setEaseInOutQuart().s
etIgnoreTimeScale(true);;
        yield return tenthSecondDelay;
        Menu.ins.carName.text = Menu.ins.cars[Menu.ins.selectedCar].gameObject
.name;
        Menu.ins.carName.gameObject.LeanScaleX(1f, 0.1f).setEaseInOutQuart().s
etIgnoreTimeScale(true);;
        yield return twoFifthsSecondDelay;
        Menu.ins.animating = false;
    }
    private void CheckCarLock()
```

```
    {
        if (Menu.ins.gameType == Menu.GameTypes.play)
        {
            if (Menu.ins.selectedCar <= 1) { PlayerPrefs.SetInt("Car" + Menu.i
ns.selectedCar.ToString(), 1); }
            if (PlayerPrefs.GetInt("Car" + Menu.ins.selectedCar.ToString(), 0)
 == 0)
            { LockCar(); }
            else
            { StartCoroutine(UnlockCar()); }
        }
        else
        { StartCoroutine(UnlockCar()); }
    }
    private void LockCar()
    {
        Menu.ins.playButtonCar.interactable = false;
        Menu.ins.carLockText.text = Menu.ins.carLockTexts[Menu.ins.selectedCar
];
        carLockObject.SetActive(true);
        carLockObject.LeanScale(new Vector3(1f, 1f, 1f), 0.25f).setEaseInOutQu
art().setIgnoreTimeScale(true);;
    }
    private IEnumerator UnlockCar()
    {
        Menu.ins.playButtonCar.interactable = true;
        carLockObject.LeanScale(new Vector3(0f, 0f, 0f), 0.25f).setEaseInOutQu
art().setIgnoreTimeScale(true);;
        yield return quarterSecondDelay;
        carLockObject.SetActive(false);
        Menu.ins.carLockText.text = string.Empty;
    }
}
```

5.2.15 - Script for Quick Race Settings Screen (QuickRaceSettings.cs)

This component is also a child of the Menu class. It is activated only if the "Quick Race" game type is selected. The player can toggle through a list of available race settings.

```
public class QuickRaceSettings : MonoBehaviour
{
    private IEnumerator SetQuickRaceSettings()
    {
        if (Menu.ins.gameType == Menu.GameTypes.quickRace2P)
        {
            if (Menu.ins.raceSettings1Car.isOn) { Menu.ins.raceSettings2Cars.i
sOn = true; }
            Menu.ins.raceSettings1Car.interactable = false;
```

```
            Menu.ins.backButtonSettingsQuickRace2P.gameObject.SetActive(true);
            Menu.ins.backButtonSettingsQuickRace1P.gameObject.SetActive(false)
;
        }
        else
        {
            Menu.ins.raceSettings1Car.interactable = true;
            Menu.ins.backButtonSettingsQuickRace2P.gameObject.SetActive(false)
;
            Menu.ins.backButtonSettingsQuickRace1P.gameObject.SetActive(true);
        }
        Menu.ins.FadeIn(Menu.ins.raceSettingsCanvas);
        Menu.ins.carsParent.SetActive(false);
        if (Menu.ins.menuScreen == Menu.MenuScreens.levels)
        {
            Menu.ins.SetRaceSettings();
            Menu.ins.menuScreen = Menu.MenuScreens.settings;
            Menu.ins.mainCars.SetActive(false);
            StartCoroutine(Menu.ins.FadeOut(Menu.ins.levelSelectionCanvas));
        }
        else if (Menu.ins.menuScreen == Menu.MenuScreens.cars)
        {
            Menu.ins.menuScreen = Menu.MenuScreens.settings;
            StartCoroutine(Menu.ins.FadeOut(Menu.ins.carSelectionCanvas));
            Menu.ins.mainCamera.LeanMove(Menu.ins.cameraPosition, Menu.ins.hal
fSecond).setEaseInOutQuart().setIgnoreTimeScale(true);
            Menu.ins.mainCamera.LeanRotate(Menu.ins.cameraRotation, Menu.ins.h
alfSecond).setEaseInOutQuart().setIgnoreTimeScale(true);
            yield return new WaitForSecondsRealtime(0.5f);
            Menu.ins.cameraAnimation.enabled = true;
        }
        if (Menu.ins.carsInstantiatedB)
        { Menu.ins.carsB[Menu.ins.selectedCarB].rotation = Quaternion.Euler(0f
, -90f, 0f); }
    }
}
```

5.2.16 - Script for Opening and Closing Menu Windows (MenuWindows.cs)

This component controls the windows activated when the buttons on the bottom bar of the main menu is clicked. It fades in or out the window and changes the camera's depth of field to create a blur effect on the background.

```
public class MenuWindows : MonoBehaviour
{
    public void OpenWindow(CanvasGroup windowCanvas)
    {
        Menu.ins.windowOpen = true;
```

```
        Menu.ins.animating = true;
        if (Menu.ins.postProcessor.profile.TryGetSettings(out Menu.ins.depthOf
Field))
        { LeanTween.value(3.5f, 1f, Menu.ins.halfSecond).setEaseInOutQuart().s
etIgnoreTimeScale(true).setOnUpdate((float depth) => { Menu.ins.depthOfField.f
ocusDistance.value = depth; }); }
        Menu.ins.FadeIn(windowCanvas);
        StartCoroutine(Menu.ins.FadeOut(Menu.ins.mainMenuCanvas));
    }
    public void CloseWindow(CanvasGroup windowCanvas)
    {
        Menu.ins.animating = true;
        if (Menu.ins.postProcessor.profile.TryGetSettings(out Menu.ins.depthOf
Field))
        { LeanTween.value(1f, 3.5f, Menu.ins.halfSecond).setEaseInOutQuart().s
etIgnoreTimeScale(true).setOnUpdate((float depth) => { Menu.ins.depthOfField.f
ocusDistance.value = depth; }); }
        Menu.ins.FadeIn(Menu.ins.mainMenuCanvas);
        StartCoroutine(Menu.ins.FadeOut(windowCanvas));
        Menu.ins.windowOpen = false;
    }
}
```

5.3 - Test Plan and Scenarios

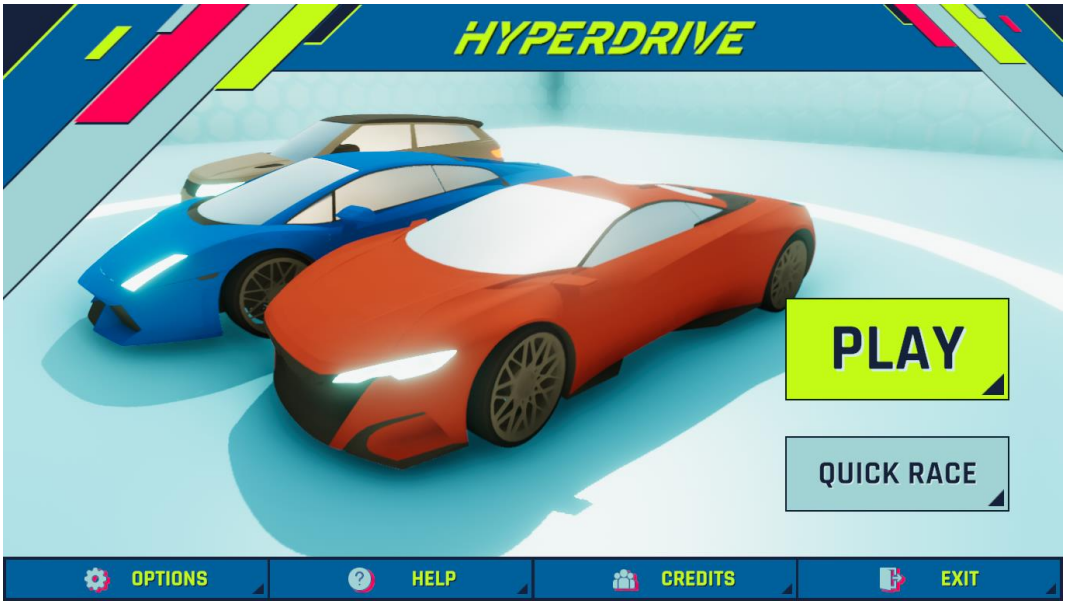
Test No.	Component Tested	Purpose of Test	Observation	Result	Test Date
1	Main menu (Unit test)	To test whether the player can interact with all the buttons and the desired screen is displayed	The player can select the Play, Quick Race, help, credits, options and exit buttons and the correct screen is displayed	PASS	05.05.21
2	Play game (Module test)	To test whether the level is loaded when the player clicks the play button	The race scene is loaded with all the aspects and the player can play the game	PASS	05.05.21
3	Selected car (Module test)	To test if the car that the player has selected in menu is the one they are controlling in the race	The player has got the car that they have selected in the menu to race	PASS	05.05.21

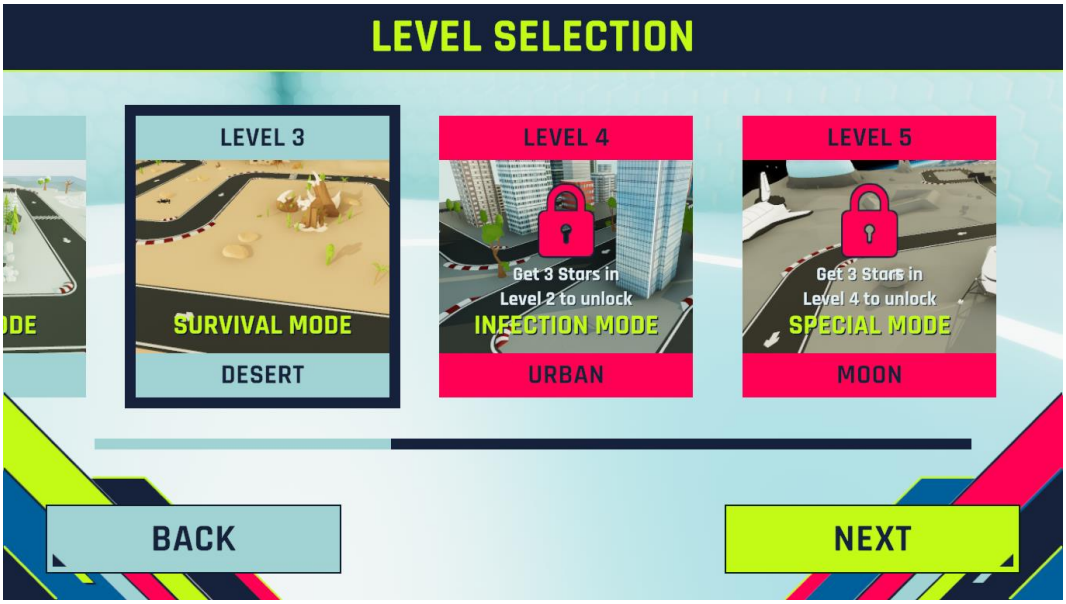
4	Car control (Unit test)	To test if the player can control the cars with Arrow keys and WASD	The player can control the car with the arrow keys and WASD effortlessly	PASS	05.05.21
5	Car collision (Unit test)	To test if there are any glitch when the car collides with other cars	No glitches were found when there was a collision between cars	PASS	06.05.21
6	2-player mode (System test)	To test if the screen is being split and each player can control his/her car	The screen was split and each player was able to control his/her respective car	PASS	06.05.21
7	Star ratings (Module test)	To test if the player obtains 3 stars if they win the race	The player has been granted 3 stars after winning the race	PASS	06.05.21
8	Escape mode (Unit test)	To test if the player is eliminated if they did not finish the lap after the timer is over	The player is eliminated	PASS	07.05.21
9	Survival mode (Unit test)	To test if the player is eliminated if they are in the last place after the timer is over	The player is eliminated	PASS	07.05.21
10	Infection mode (Unit test)	To test if the player is infected if they are in the last place after the timer is over	The player is infected and got an engine boost	PASS	07.05.21
11	Moon map (Module test)	To test if the gravity is lowered in the Moon map	The gravity is lowered	PASS	07.05.21
12	Rank (System test)	To test is the rank changes in real-time	The rank changes when overtaking cars	PASS	07.05.21

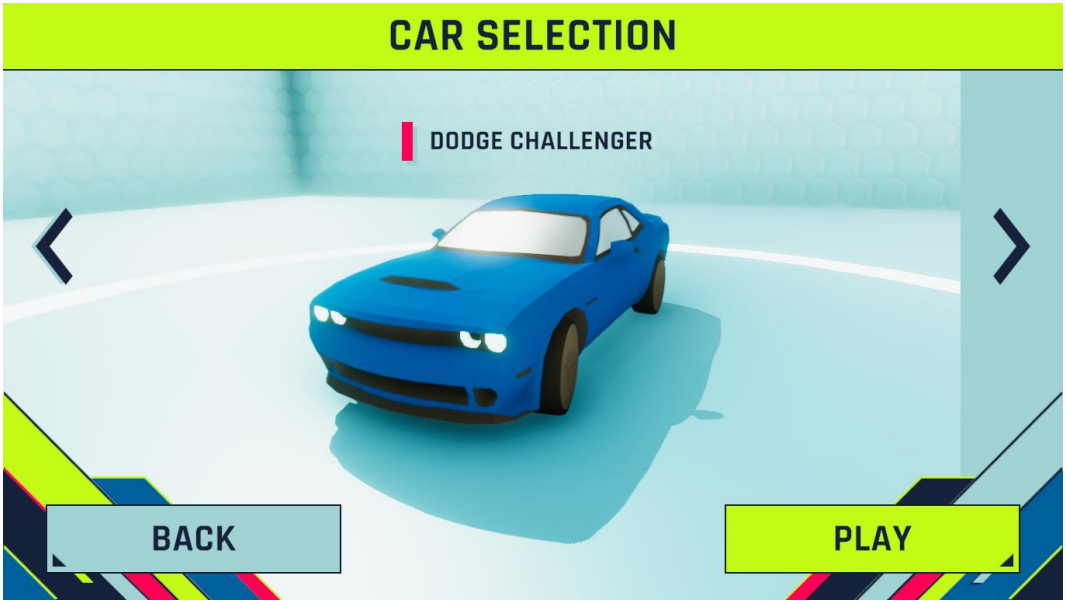
13	Item unlock (Module test)	To test if the appropriate new item is unlocked when obtaining 3 stars	New car was unlocked when winning Level 1	PASS	08.05.21
14	Race leader board (Unit test)	To test if the leader board if correctly displayed after the race is over	The leader board is displayed in the Unity Editor but not in the built application	FAIL	08.05.21
15	Music (System test)	To test if the music changes when the Next and Previous Song button is clicked	The song is changed	PASS	09.05.21
16	Sound (Unit test)	To test if the volume changes correctly	The volume changes when the slider is adjusted	PASS	10.05.21
17	Game progress (System test)	To test if the game progress can be reset	The progress is cleared when the Reset button is clicked	PASS	11.05.21
18	Reset car position (Unit test)	To test if the position of the car is reset to the last checkpoint	The car's position resets when Enter/Tab is pressed and when falling off the map	PASS	12.05.21
19	Restart race (Unit test)	To test if the race can be restarted	The scene reloads correctly	PASS	13.05.21
20	Exit game (Unit test)	To test if the game can be exited	The game closes	PASS	14.05.21

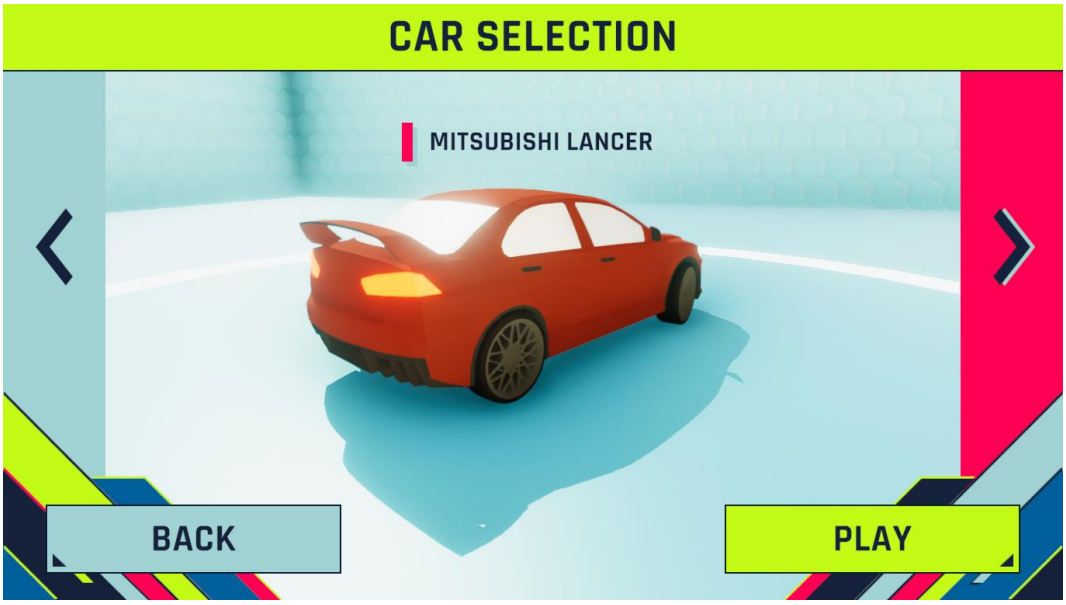
Table 2 - Test Plan

5.4 - Sample Screenshots



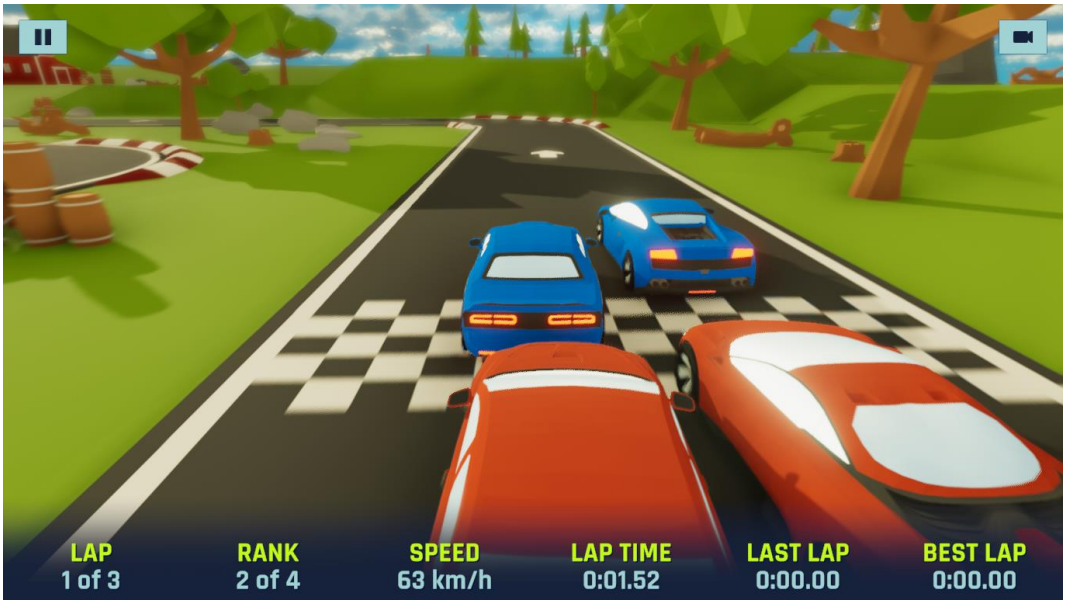

LEVEL SELECTION
LEVEL 3
SURVIVAL MODE
DESERT
LEVEL 4
Get 3 Stars in
Level 2 to unlock
INFECTION MODE
URBAN
LEVEL 5
Get 3 Stars in
Level 4 to unlock
SPECIAL MODE
MOON
BACK
NEXT

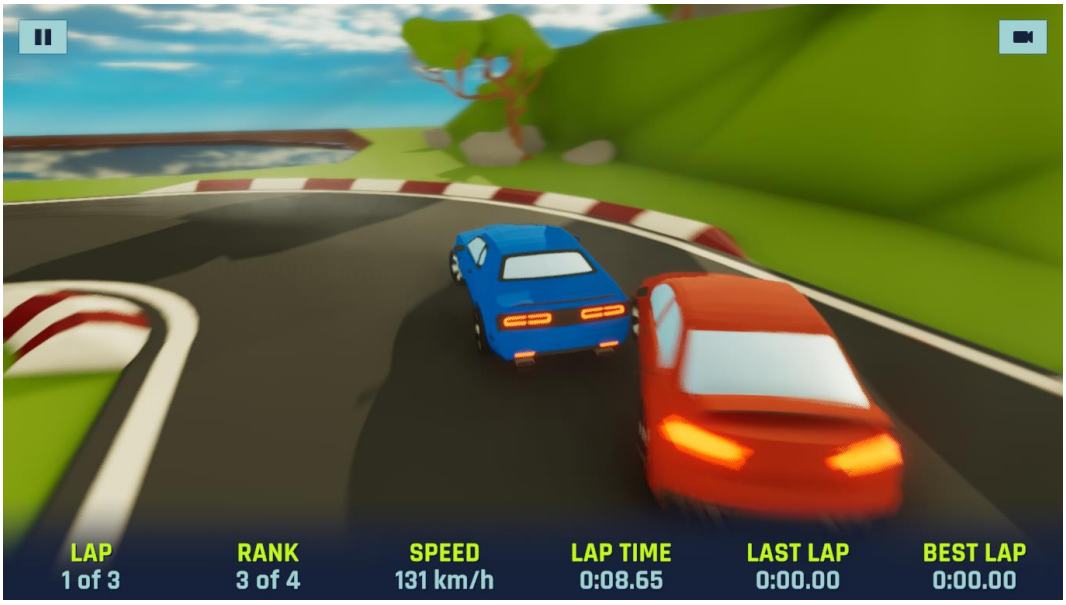

CAR SELECTION
DODGE CHALLENGER
BACK
PLAY

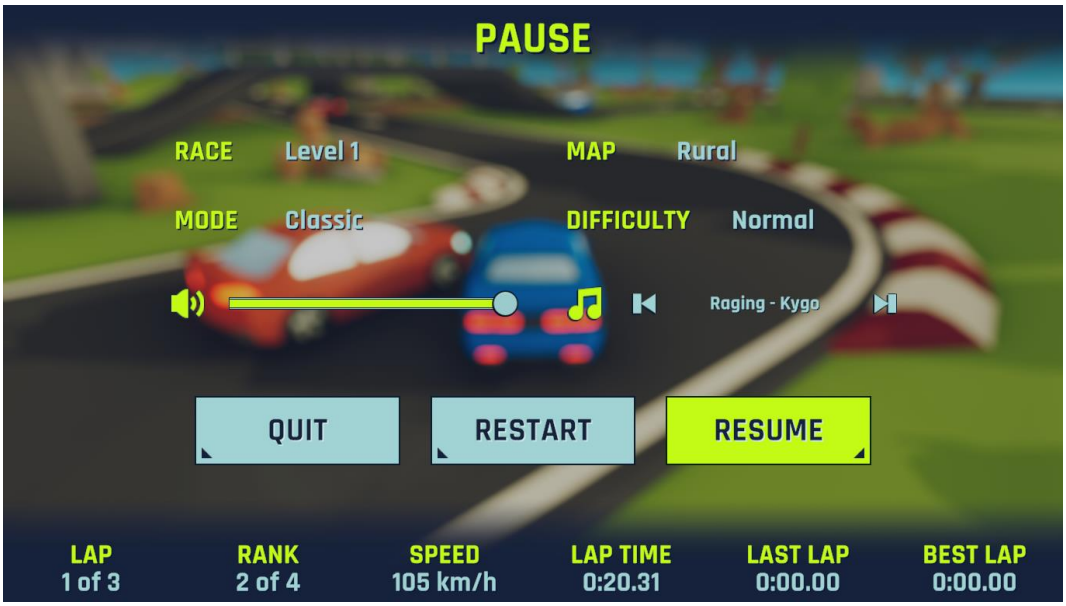

CAR SELECTION
MITSUBISHI LANCER
BACK
PLAY


CAR SELECTION
LAMBORGHINI HURACAN
Get 3 stars in
Level 5 to unlock
BACK
PLAY

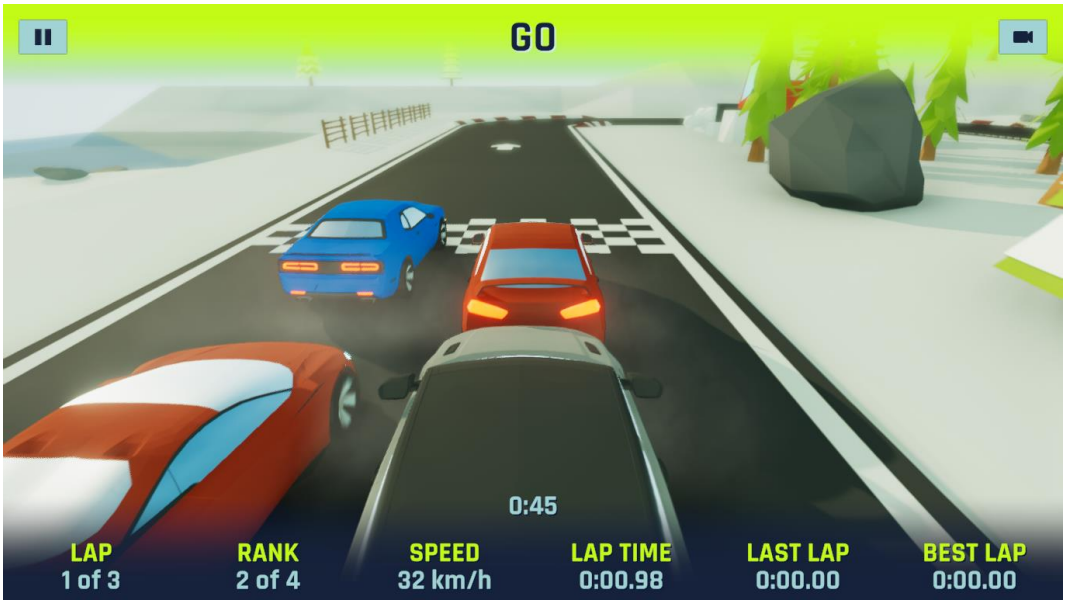

3


LAP
1 of 3
RANK
2 of 4
SPEED
63 km/h
LAP TIME
0:01.52
LAST LAP
0:00.00
BEST LAP
0:00.00

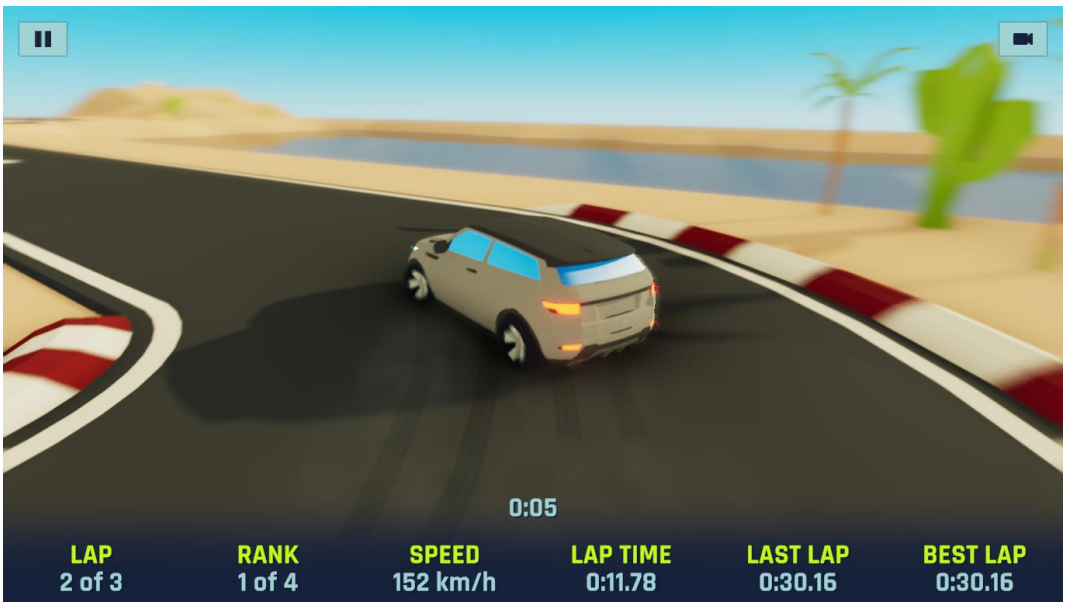

LAP
1 of 3
RANK
3 of 4
SPEED
131 km/h
LAP TIME
0:08.65
LAST LAP
0:00.00
BEST LAP
0:00.00

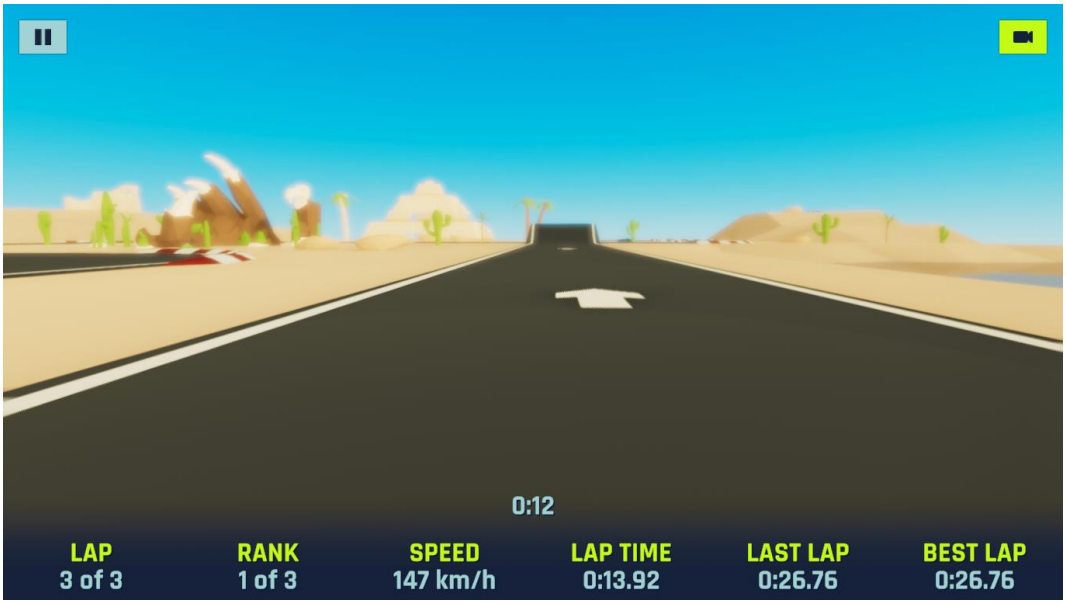

PAUSE
RACE
Level 1
MAP
Rural
MODE
Classic
DIFFICULTY
Normal
Raging - Kygo
QUIT
RESTART
RESUME
LAP
1 of 3
RANK
2 of 4
SPEED
105 km/h
LAP TIME
0:20.31
LAST LAP
0:00.00
BEST LAP
0:00.00

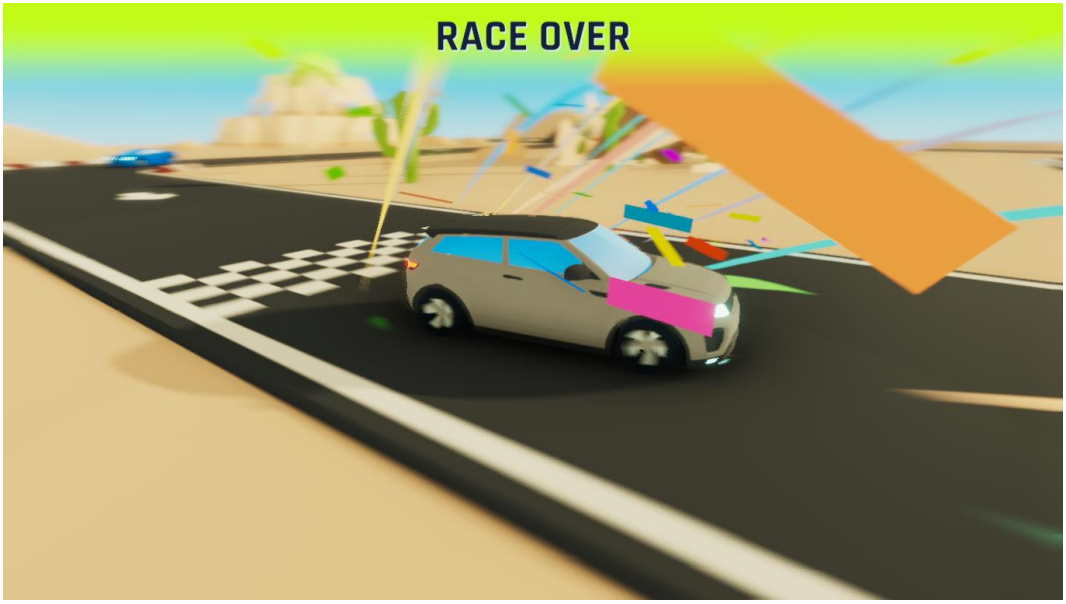

RACE OVER
RANK
1
RACE
Level 1
MAP
Rural
MODE
Classic
LAPS
3
DIFFICULTY
Normal
RESTART
NEW CAR UNLOCKED
RANGE ROVER EVOQUE
NEXT

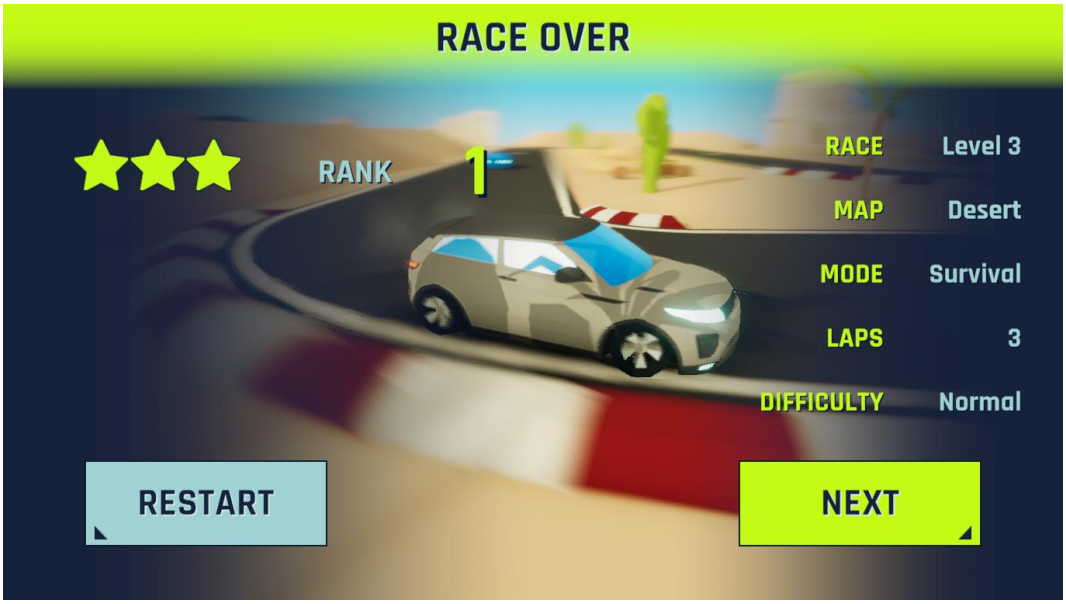

GO
0:45
LAP
1 of 3
RANK
2 of 4
SPEED
32 km/h
LAP TIME
0:00.98
LAST LAP
0:00.00
BEST LAP
0:00.00


0:40
LAP
1 of 3
RANK
3 of 4
SPEED
105 km/h
LAP TIME
0:05.45
LAST LAP
0:00.00
BEST LAP
0:00.00

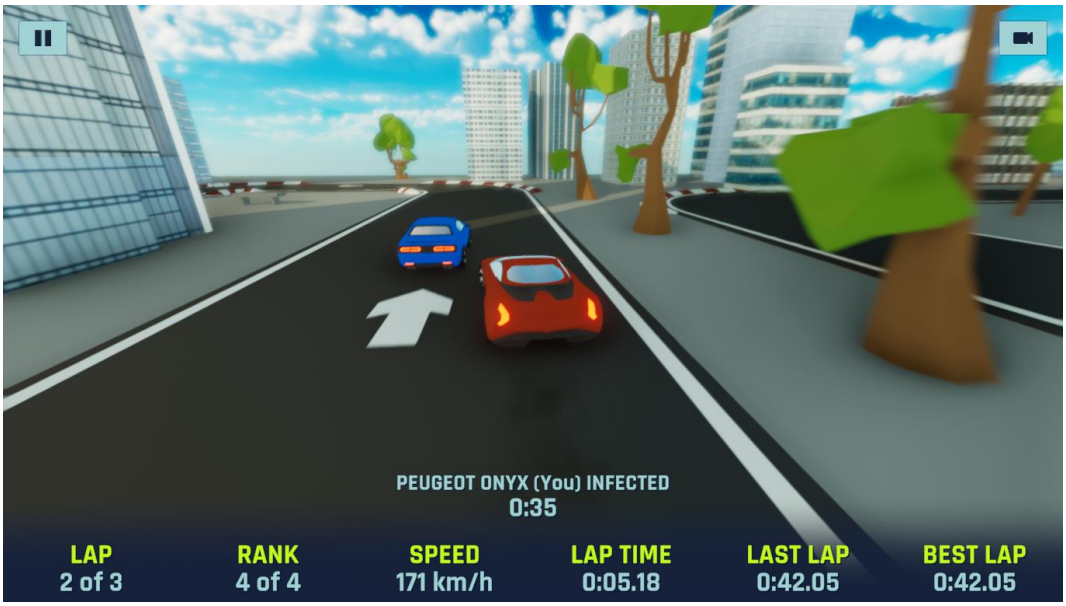

0:05
LAP
2 of 3
RANK
1 of 4
SPEED
152 km/h
LAP TIME
0:11.78
LAST LAP
0:30.16
BEST LAP
0:30.16


0:12
LAP
3 of 3
RANK
1 of 3
SPEED
147 km/h
LAP TIME
0:13.92
LAST LAP
0:26.76
BEST LAP
0:26.76


RACE OVER

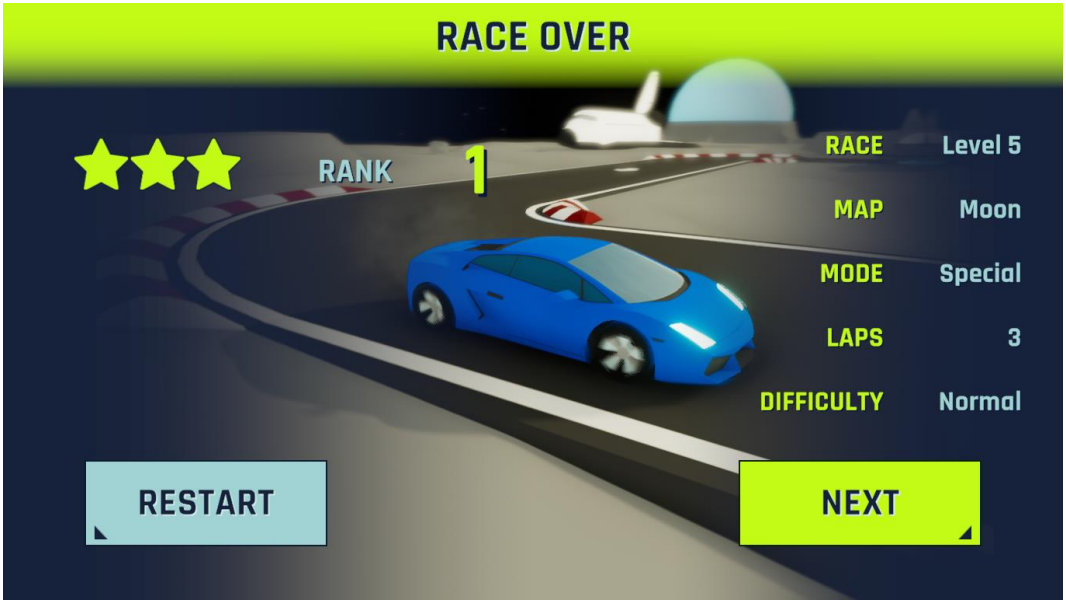

RACE OVER
RANK
1
RACE
Level 3
MAP
Desert
MODE
Survival
LAPS
3
DIFFICULTY
Normal
RESTART
NEXT


MITSUBISHI LANCER INFECTED
0:25
LAP
3 of 3
RANK
1 of 4
SPEED
129 km/h
LAP TIME
0:24.20
LAST LAP
0:26.36
BEST LAP
0:26.36


PEUGEOT ONYX (You) INFECTED
0:35
LAP
2 of 3
RANK
4 of 4
SPEED
171 km/h
LAP TIME
0:05.18
LAST LAP
0:42.05
BEST LAP
0:42.05

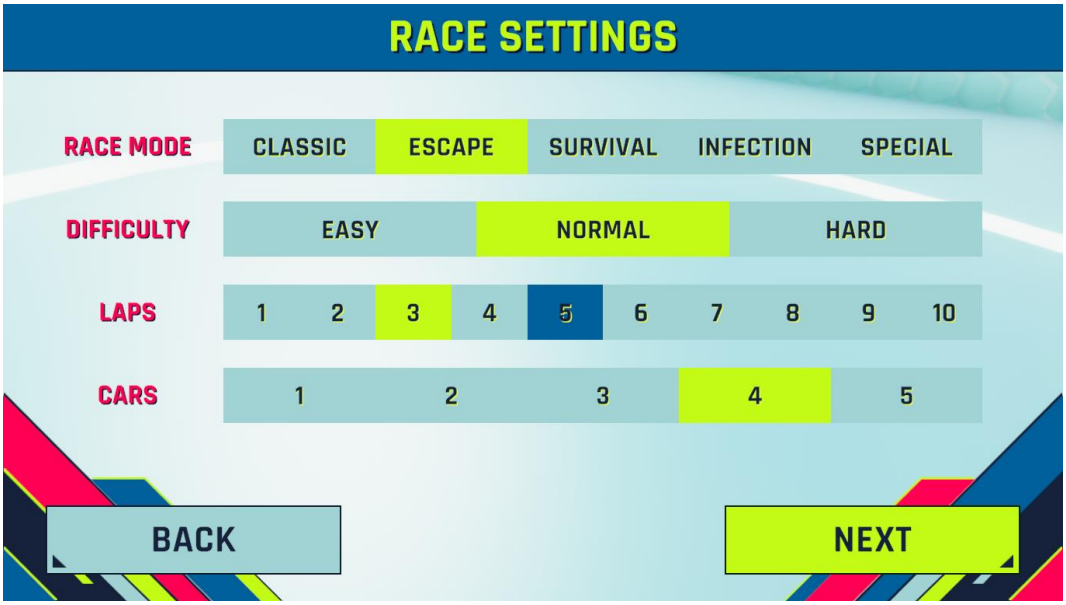

0:06
LAP
1 of 3
RANK
1 of 4
SPEED
102 km/h
LAP TIME
0:40.87
LAST LAP
0:00.00
BEST LAP
0:00.00


0:28
LAP
1 of 3
RANK
2 of 4
SPEED
133 km/h
LAP TIME
0:17.98
LAST LAP
0:00.00
BEST LAP
0:00.00

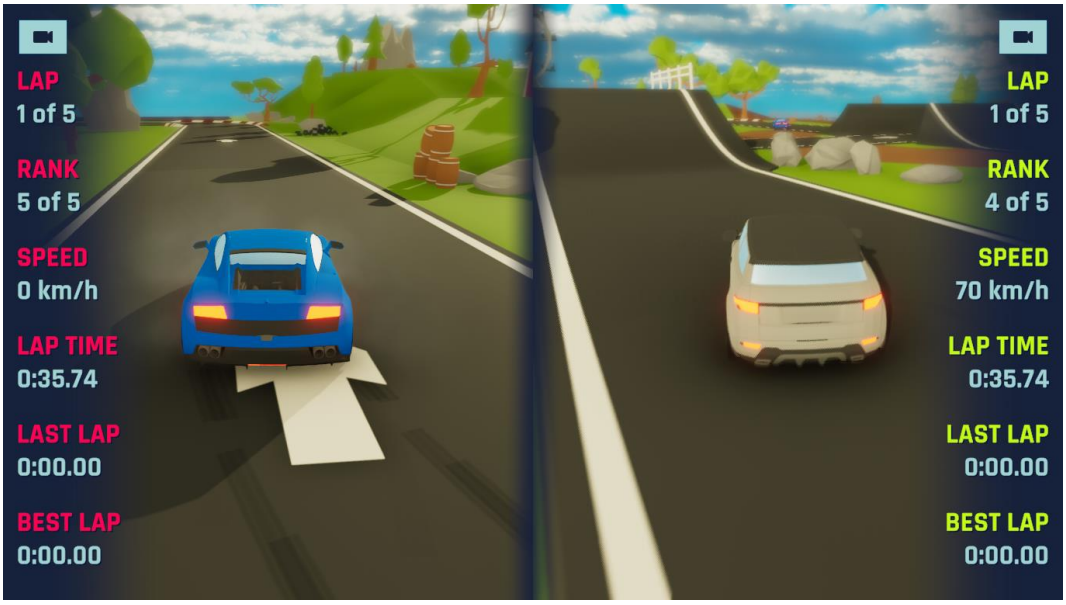

RACE OVER
RANK
1
RACE
Level 5
MAP
Moon
MODE
Special
LAPS
3
DIFFICULTY
Normal
RESTART
NEXT


LEVEL SELECTION
LEVEL 1
CLASSIC MODE
RURAL
LEVEL 2
ESCAPE MODE
ARCTIC
LEVEL 3
SURVIVAL MODE
DESERT
BACK
NEXT

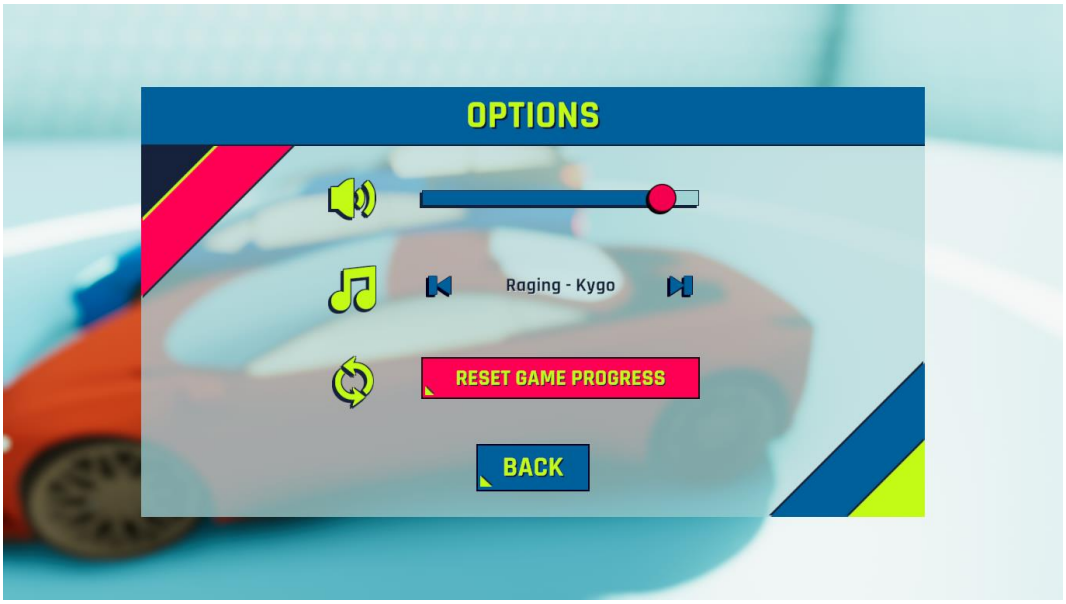

QUICK RACE
1 PLAYER
2 PLAYERS
BACK


RACE SETTINGS
RACE MODE
CLASSIC
ESCAPE
SURVIVAL
INFECTION
SPECIAL
DIFFICULTY
EASY
NORMAL
HARD
LAPS
1
2
3
4
5
6
7
8
9
10
CARS
1
2
3
4
5
BACK
NEXT


GO
LAP
1 of 5
RANK
2 of 5
SPEED
38 km/h
LAP TIME
0:01.10
LAST LAP
0:00.00
BEST LAP
0:00.00
LAP
1 of 5
RANK
3 of 5
SPEED
44 km/h
LAP TIME
0:01.10
LAST LAP
0:00.00
BEST LAP
0:00.00


LAP
1 of 5
RANK
5 of 5
SPEED
0 km/h
LAP TIME
0:35.74
LAST LAP
0:00.00
BEST LAP
0:00.00
LAP
1 of 5
RANK
4 of 5
SPEED
70 km/h
LAP TIME
0:35.74
LAST LAP
0:00.00
BEST LAP
0:00.00


LAP
1 of 5
RANK
5 of 5
SPEED
0 km/h
LAP TIME
0:36.76
LAST LAP
0:00.00
BEST LAP
0:00.00
QUIT RACE?
Are you sure you want
to quit the race?
YES
NO
LAP
1 of 5
RANK
4 of 5
SPEED
94 km/h
LAP TIME
0:36.76
LAST LAP
0:00.00
BEST LAP
0:00.00


OPTIONS
Raging - Kygo
RESET GAME PROGRESS
BACK

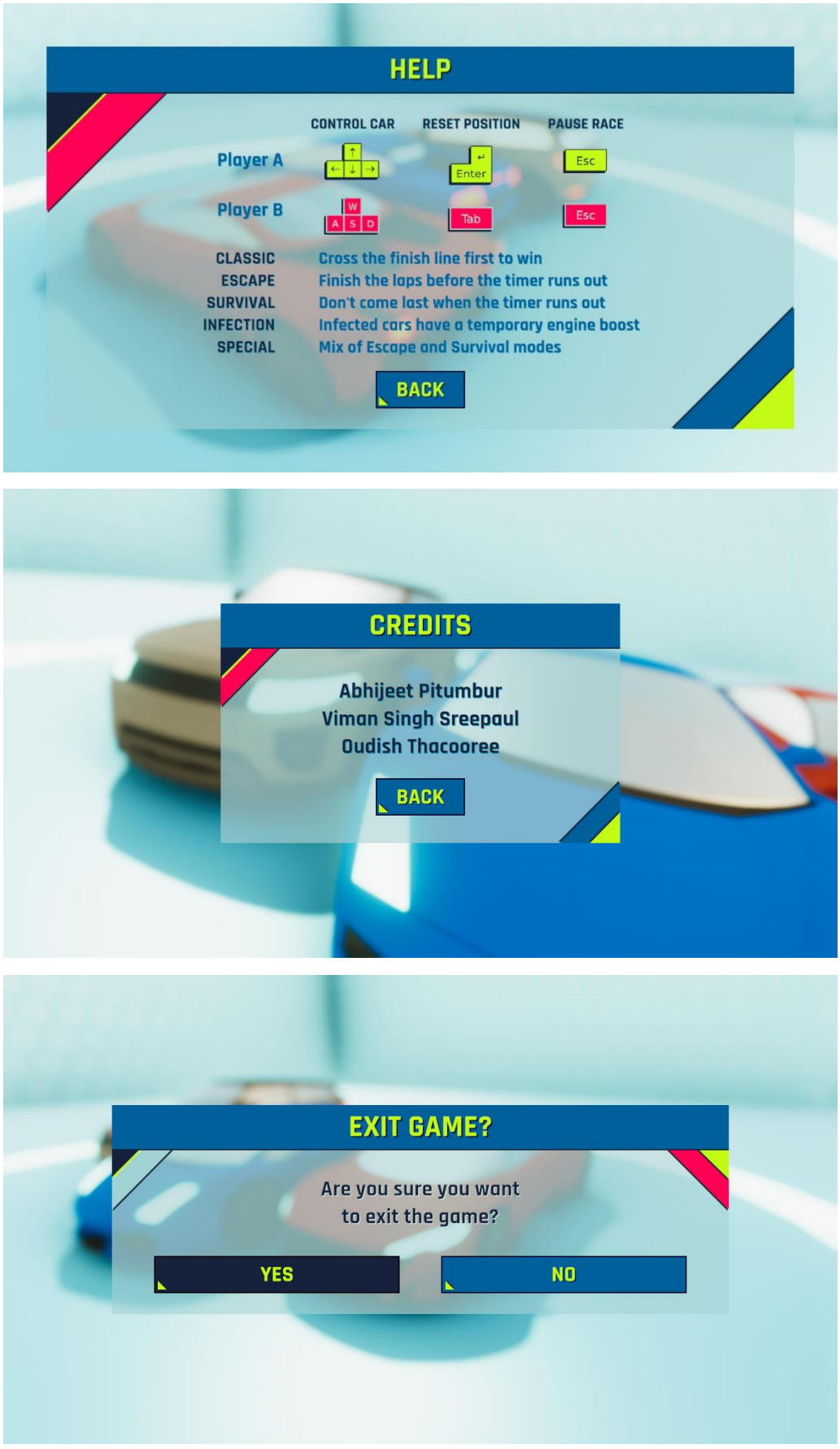


Figure 25 - Hyperdrive Screenshots

6 - Conclusion

6.1 - Achievements

6.1.1 - Technical Side

- Learned the C# language.
- Learned more about Object-Oriented Programming.
- Understood how the Unity Game Engine works
- Learned how to build game assets on Blender.
- Learned how to use those assets.
- Learned how to add sounds into our game.
- Obtained the ability to work together as a team.
- Learned how to design a game menu.

6.1.2 - Non-Technical Side

- Improved our communication skills.
- Learned how to analyse and solve different problems.
- Improved our report writing skills.
- Developed self-learning skills.
- We now work better under pressure.
- Learned about proper work planning.
- Improved our time management skills.

6.2 - Challenges and Problems Encountered

- Unity was hard to learn and master.
- Lots of time was used up to detect and correct bugs in the game.
- Lots of time had to be invested in the development of the game.
- Blender shortcuts and features were hard to learn.
- Importing assets from blender to unity was complicated.
- Learning the C# language posed some problems despite the latter resembling C++.
- While developing the game, many crashes occurred both in Unity and Blender.

6.3 - Future Work

We have been working for quite some time on both Unity and Blender for a while now. Add to that our newfound knowledge in C#, we will be able to improve our game in general using our knowledge and add new features such as:

- New levels
- New cars
- Adding different stats to the cars
- Making the game online
- New race modes
- Make the game playable on mobile phones

7 - Appendix

7.1.1 - User Manual for Main Menu

- Play – To play the career mode.
- Quick Race – To play a quick race or local multiplayer.
- Options – To change sound settings.
- Reset Game – To reset the game.
- Help – Teaches the controls and explains game modes.
- Credits – Shows the names of the game developers.
- Exit – To exit the game.

7.1.2 - User Manual for Driving the Car (Player A)

- Up arrow key – Drive forward
- Left arrow key – Steer left
- Right arrow key – Steer right
- Down arrow key – Reverse/brake
- Enter – Reset car position to last checkpoint

7.1.3 - User Manual for Driving the Car (Player B)

- W key – Drive forward
- A key – Steer left
- D key – Steer right
- S key – Reverse/brake
- Tab – Reset car position to last checkpoint

7.1.4 - User Manual for Pause Menu

- Resume – To resume the race.
- Quit – To quit the race and go to Main Menu.
- Restart – To restart the race.

8 - REFERENCES

[1] Brackeys. (2017, December 6). SETTINGS MENU in Unity. YouTube. https://www.youtube.com/watch?v=YOaYQrN1oYQ

[2] Code Monkey. (2020, July 20). Learn Unity in 17 MINUTES! YouTube. https://www.youtube.com/watch?v=E6A4WvsDeLE

[3] Code Monkey. (2020b, December 23). Simple Checkpoint System in Unity. YouTube. https://www.youtube.com/watch?v=IOYNg6v9sfc

[4] Code Monkey. (2021, January 20). Simple Car AI Driver in Unity! YouTube. https://www.youtube.com/watch?v=xuCtxIcfboM

[5] Dolan, H. (2021, April 28). The Ultimate Beginners Guide To Game Development In Unity. FreeCodeCamp.Org. https://www.freecodecamp.org/news/the-ultimate-beginners-guide-to-game-development-in-unity-f9bfe972c2b5/

[6] Imphenzia. (2020, July 24). LEARN UNITY - The Most BASIC TUTORIAL I'll Ever Make. YouTube. https://www.youtube.com/watch?v=pwZpJzpE2lQ

[7] Imphenzia. (2020a, June 25). Learn Low Poly Modeling in Blender 2.9 / 2.8. YouTube. https://www.youtube.com/watch?v=1jHUY3qoBu8

[8] Imphenzia. (2020a, March 25). Low Poly Racing - Making Of - Ep 1 - Unity & Blender. YouTube. https://www.youtube.com/watch?v=ODVV3eUE5zM

[9] Jimmy Vegas. (2017, July 13). How To Make A Driving & Racing Game For FREE - Unity Tutorial #07 - AI & WAYPOINT SYSTEM. YouTube. https://www.youtube.com/watch?v=TdNbxkF_f94

[10] Karting Microgame. (2021). Unity Learn. https://learn.unity.com/project/karting-template

[11] Making a race car positioning system - Unity Answers. (2021). Unity Answers. https://answers.unity.com/questions/845447/making-a-race-car-positioning-system.html

[12] N. (2020, May 20). Easy Split-Screen Multiplayer in Unity 3D. Sharp Coder Blog. https://sharpcoderblog.com/blog/easy-split-screen-multiplayer-in-unity-3d

[13] SpeedTutor. (2018, July 24). SIMPLE PAUSE MENU in Unity. YouTube. https://www.youtube.com/watch?v=pbeB9NsaoPs

END OF DOCUMENT

Abhijeet Pitumbur, Viman Singh Sreepaul, Oudish Thacooree © 2021